

CS 4110

# Programming Languages & Logics

Victory Lap!



# Victory Lap

We've covered a *lot* of territory this semester...

...now you're ready to soar!



# Semester in Review

[CS 4110](#) [Home](#) [Project](#) [Resources](#) [Schedule](#) [Syllabus](#) [CMS](#) [Campuswire](#)

## Schedule

Date	Topic	Notes
September 2	Course Overview	<a href="#">slides</a> <a href="#">video</a>
September 4	Introduction to Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 7	Inductive Definitions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 9	Properties and Inductive Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>
September 11	Inductive Proof and Large-Step Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 14	The IMP Language	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 16	IMP Properties	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 18	More IMP Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">whiteboard-1</a> <a href="#">whiteboard-2</a> <a href="#">video-1</a> <a href="#">video-2</a>
September 21	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 23	Denotational Semantics Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 25	Axiomatic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 28	Hoare Logic	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 30	Hoare Logic Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 2	Weakest Preconditions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 5	$\lambda$ -Calculus	<a href="#">notes</a> <a href="#">video</a>
October 7	More $\lambda$ -Calculus and Substitution	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 9	de Bruijn and Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 12	Encodings	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 14	<b>Mid-semester break</b>	

October 16	Encodings and Fixed-Point Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 19	Definitional Translation	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW4 due
October 21	Continuations	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 23	Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 26	More Types	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	HW5 due
October 28	Proving Type Soundness	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 30	More Proving Type Soundness	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	
November 2	Normalization	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW6 due
November 4	Advanced Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 6	Polymorphism	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 9	Type Inference	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Take-Home Prelim 2
November 11	Recursive Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 13	Records and Subtyping	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Project Alpha due
November 16	<b>Study days</b>		
November 18	<b>Semi-finals</b>		
November 20	<b>Semi-finals</b>		
November 23	<b>Semi-finals</b>		
November 25	<b>No class (Thanksgiving)</b>		
November 27	<b>No class (Thanksgiving)</b>		
November 30	Existential Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 2	Propositions as Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 4	Dependent Types	<a href="#">notes</a> <a href="#">video</a> <a href="#">code</a>	Project Beta due
December 7	Concurrency	<a href="#">slides</a> <a href="#">video</a>	Homework 7 due
December 9	Probabilistic Semantics		
December 11	After 4110	<a href="#">slides</a>	
December 14	DSLs and Bidirectional Programming	<a href="#">slides</a> <a href="#">video</a>	Homework 8 due
December 16	Victory Lap		Final Project due

# Semester in Review

CS 4110 Home Project Resources Schedule Syllabus CMS Campuswire

## Schedule

Date	Topic	Notes
September 2	Course Overview	<a href="#">slides</a> <a href="#">video</a>
September 4	Introduction to Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 7	Inductive Definitions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 9	Properties and Inductive Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>
September 11	Inductive Proof and Large-Step Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 14	The IMP Language	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 16	IMP Properties	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 18	More IMP Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">whiteboard-1</a> <a href="#">whiteboard-2</a> <a href="#">video-1</a> <a href="#">video-2</a>
September 21	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 23	Denotational Semantics Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 25	Axiomatic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 28	Hoare Logic	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 30	Hoare Logic Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 2	Weakest Preconditions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 5	$\lambda$ -Calculus	<a href="#">notes</a> <a href="#">video</a>
October 7	More $\lambda$ -Calculus and Substitution	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 9	de Bruijn and Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 12	Encodings	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 14	Mid-semester break	

## Mathematical Preliminaries

October 16	Encodings and Fixed-Point Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 19	Definitional Translation	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW4 due
October 21	Continuations	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 23	Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 26	More Types	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	HW5 due
October 28	Proving Type Soundness	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 30	More Proving Type Soundness	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	
November 2	Normalization	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW6 due
November 4	Advanced Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 6	Polymorphism	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 9	Type Inference	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Take-Home Prelim 2
November 11	Recursive Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 13	Records and Subtyping	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Project Alpha due
November 16	<b>Study days</b>		
November 18	<b>Semi-finals</b>		
November 20	<b>Semi-finals</b>		
November 23	<b>Semi-finals</b>		
November 25	<b>No class (Thanksgiving)</b>		
November 27	<b>No class (Thanksgiving)</b>		
November 30	Existential Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 2	Propositions as Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 4	Dependent Types	<a href="#">notes</a> <a href="#">video</a> <a href="#">code</a>	Project Beta due
December 7	Concurrency	<a href="#">slides</a> <a href="#">video</a>	Homework 7 due
December 9	Probabilistic Semantics		
December 11	After 4110	<a href="#">slides</a>	
December 14	DSLs and Bidirectional Programming	<a href="#">slides</a> <a href="#">video</a>	Homework 8 due
December 16	Victory Lap		Final Project due

# Semester in Review

CS 4110 Home Project Resources Schedule Syllabus CMS Campuswire

## Schedule

Date	Topic	Notes
September 2	Course Overview	<a href="#">slides</a> <a href="#">video</a>
September 4	Introduction to Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 7	Inductive Definitions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 9	Properties and Inductive Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>
September 11	Inductive Proof and Large-Step Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 14	The IMP Language	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 16	IMP Properties	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 18	More IMP Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">whiteboard-1</a> <a href="#">whiteboard-2</a> <a href="#">video-1</a> <a href="#">video-2</a>
September 21	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 23	Denotational Semantics Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 25	Axiomatic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 28	Hoare Logic	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 30	Hoare Logic Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 2	Weakest Preconditions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 5	$\lambda$ -Calculus	<a href="#">notes</a> <a href="#">video</a>
October 7	More $\lambda$ -Calculus and Substitution	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 9	de Bruijn and Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 12	Encodings	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 14	<b>Mid-semester break</b>	

Mathematical Preliminaries

Operational Semantics

October 16	Encodings and Fixed-Point Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 19	Definitional Translation	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW4 due
October 21	Continuations	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 23	Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 26	More Types	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	HW5 due
October 28	Proving Type Soundness	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 30	More Proving Type Soundness	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	
November 2	Normalization	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW6 due
November 4	Advanced Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 6	Polymorphism	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 9	Type Inference	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Take-Home Prelim 2
November 11	Recursive Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 13	Records and Subtyping	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Project Alpha due
November 16	<b>Study days</b>		
November 18	<b>Semi-finals</b>		
November 20	<b>Semi-finals</b>		
November 23	<b>Semi-finals</b>		
November 25	<b>No class (Thanksgiving)</b>		
November 27	<b>No class (Thanksgiving)</b>		
November 30	Existential Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 2	Propositions as Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 4	Dependent Types	<a href="#">notes</a> <a href="#">video</a> <a href="#">code</a>	Project Beta due
December 7	Concurrency	<a href="#">slides</a> <a href="#">video</a>	Homework 7 due
December 9	Probabilistic Semantics		
December 11	After 4110	<a href="#">slides</a>	
December 14	DSLs and Bidirectional Programming	<a href="#">slides</a> <a href="#">video</a>	Homework 8 due
December 16	Victory Lap		Final Project due

# Semester in Review

## Schedule

Date	Topic	Notes
September 2	Course Overview	<a href="#">slides</a> <a href="#">video</a>
September 4	Introduction to Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 7	Inductive Definitions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 9	Properties and Inductive Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>
September 11	Inductive Proof and Large-Step Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 14	The IMP Language	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 16	IMP Properties	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 18	More IMP Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">whiteboard-1</a> <a href="#">whiteboard-2</a> <a href="#">video-1</a> <a href="#">video-2</a>
September 21	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 23	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 25	Axiomatic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 28	Hoare Logic	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 30	Hoare Logic Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 2	Weakest Preconditions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 5	$\lambda$ -Calculus	<a href="#">notes</a> <a href="#">video</a>
October 7	More $\lambda$ -Calculus and Substitution	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 9	de Bruijn and Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 12	Encodings	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 14	<b>Mid-semester break</b>	

Mathematical Preliminaries

Operational Semantics

Denotational Semantics

October 16	Encodings and Fixed-Point Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 19	Definitional Translation	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW4 due
October 21	Continuations	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 23	Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 26	More Types	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	HW5 due
October 28	Proving Type Soundness	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 30	More Proving Type Soundness	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	
November 2	Normalization	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW6 due
November 4	Advanced Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 6	Polymorphism	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 9	Type Inference	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Take-Home Prelim 2
November 11	Recursive Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 13	Records and Subtyping	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Project Alpha due
November 16	<b>Study days</b>		
November 18	<b>Semi-finals</b>		
November 20	<b>Semi-finals</b>		
November 23	<b>Semi-finals</b>		
November 25	<b>No class (Thanksgiving)</b>		
November 27	<b>No class (Thanksgiving)</b>		
November 30	Existential Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 2	Propositions as Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 4	Dependent Types	<a href="#">notes</a> <a href="#">video</a> <a href="#">code</a>	Project Beta due
December 7	Concurrency	<a href="#">slides</a> <a href="#">video</a>	Homework 7 due
December 9	Probabilistic Semantics		
December 11	After 4110	<a href="#">slides</a>	
December 14	DSLs and Bidirectional Programming	<a href="#">slides</a> <a href="#">video</a>	Homework 8 due
December 16	Victory Lap		Final Project due

# Semester in Review

CS 4110 Home Project Resources Schedule Syllabus CMS Campuswire

## Schedule

Date	Topic	Notes
September 2	Course Overview	<a href="#">slides</a> <a href="#">video</a>
September 4	Introduction to Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 7	Inductive Definitions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 9	Properties and Inductive Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>
September 11	Inductive Proof and Large-Step Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 14	The IMP Language	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 16	IMP Properties	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 18	More IMP Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">whiteboard-1</a> <a href="#">whiteboard-2</a> <a href="#">video-1</a> <a href="#">video-2</a>
September 21	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 23	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 25	Axiomatic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 28	Hoare Logic	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 30	Hoare Logic Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 2	Weakest Preconditions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 5	$\lambda$ -Calculus	<a href="#">notes</a> <a href="#">video</a>
October 7	More $\lambda$ -Calculus and Substitution	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 9	de Bruijn and Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 12	Encodings	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 14	Mid-semester break	

Mathematical Preliminaries

Operational Semantics

Denotational Semantics

Axiomatic Semantics

October 16	Encodings and Fixed-Point Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 19	Definitional Translation	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW4 due
October 21	Continuations	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 23	Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 26	More Types	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	HW5 due
October 28	Proving Type Soundness	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 30	More Proving Type Soundness	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	
November 2	Normalization	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW6 due
November 4	Advanced Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 6	Polymorphism	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 9	Type Inference	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Take-Home Prelim 2
November 11	Recursive Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 13	Records and Subtyping	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Project Alpha due
November 16	<b>Study days</b>		
November 18	<b>Semi-finals</b>		
November 20	<b>Semi-finals</b>		
November 23	<b>Semi-finals</b>		
November 25	<b>No class (Thanksgiving)</b>		
November 27	<b>No class (Thanksgiving)</b>		
November 30	Existential Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 2	Propositions as Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 4	Dependent Types	<a href="#">notes</a> <a href="#">video</a> <a href="#">code</a>	Project Beta due
December 7	Concurrency	<a href="#">slides</a> <a href="#">video</a>	Homework 7 due
December 9	Probabilistic Semantics		
December 11	After 4110	<a href="#">slides</a>	
December 14	DSLs and Bidirectional Programming	<a href="#">slides</a> <a href="#">video</a>	Homework 8 due
December 16	Victory Lap		Final Project due

# Semester in Review

CS 4110 Home Project Resources Schedule Syllabus CMS Campuswire

## Schedule

Date	Topic	Notes
September 2	Course Overview	<a href="#">slides</a> <a href="#">video</a>
September 4	Introduction to Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 7	Inductive Definitions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 9	Properties and Inductive Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>
September 11	Inductive Proof and Large-Step Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 14	The IMP Language	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 16	IMP Properties	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 18	More IMP Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">whiteboard-1</a> <a href="#">whiteboard-2</a> <a href="#">video-1</a> <a href="#">video-2</a>
September 21	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 23	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 25	Axiomatic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 28	Hoare Logic	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 30	Hoare Logic Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 2	Weakest Preconditions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 5	$\lambda$ -Calculus	<a href="#">notes</a> <a href="#">video</a>
October 7	More $\lambda$ -Calculus and Substitution	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 9	de Bruijn and Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 12	Encodings	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 14	Mid-semester break	

Mathematical Preliminaries

Operational Semantics

Denotational Semantics

Axiomatic Semantics

$\lambda$ -Calculus

October 16	Encodings and Fixed-Point Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 19	Definitional Translation	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW4 due
October 21	Continuations	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 23	Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 26	More Types	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	HW5 due
October 28	Proving Type Soundness	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 30	More Proving Type Soundness	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	
November 2	Normalization	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW6 due
November 4	Advanced Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 6	Polymorphism	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 9	Type Inference	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Take-Home Prelim 2
November 11	Recursive Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 13	Records and Subtyping	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Project Alpha due
November 16	<b>Study days</b>		
November 18	<b>Semi-finals</b>		
November 20	<b>Semi-finals</b>		
November 23	<b>Semi-finals</b>		
November 25	<b>No class (Thanksgiving)</b>		
November 27	<b>No class (Thanksgiving)</b>		
November 30	Existential Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 2	Propositions as Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 4	Dependent Types	<a href="#">notes</a> <a href="#">video</a> <a href="#">code</a>	Project Beta due
December 7	Concurrency	<a href="#">slides</a> <a href="#">video</a>	Homework 7 due
December 9	Probabilistic Semantics		
December 11	After 4110	<a href="#">slides</a>	
December 14	DSLs and Bidirectional Programming	<a href="#">slides</a> <a href="#">video</a>	Homework 8 due
December 16	Victory Lap		Final Project due



# Semester in Review

## Schedule

Date	Topic	Notes
September 2	Course Overview	<a href="#">slides</a> <a href="#">video</a>
September 4	Introduction to Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 7	Inductive Definitions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 9	Properties and Inductive Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>
September 11	Inductive Proof and Large-Step Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 14	The IMP Language	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 16	IMP Properties	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 18	More IMP Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">whiteboard-1</a> <a href="#">whiteboard-2</a> <a href="#">video-1</a> <a href="#">video-2</a>
September 21	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 23	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 25	Axiomatic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 28	Hoare Logic	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 30	Hoare Logic Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 2	Weakest Preconditions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 5	$\lambda$ -Calculus	<a href="#">notes</a> <a href="#">video</a>
October 7	More $\lambda$ -Calculus and Substitution	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 9	de Bruijn and Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 12	Encodings	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 14	Mid-semester break	

Mathematical Preliminaries

Operational Semantics

Denotational Semantics

Axiomatic Semantics

$\lambda$ -Calculus

$\lambda$ -Calculus Programming

October 16	Encodings and Fixed-Point Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 19	Definitional Translation	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW4 due
October 21	Continuations	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 23	Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 26	More Types	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	HW5 due
October 28	Proving Type Soundness	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 30	More Proving Type Soundness	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	
November 2	Normalization	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW6 due
November 4	Advanced Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 6	Polymorphism	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 9	Type Inference	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Take-Home Prelim 2
November 11	Recursive Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 13	Records and Subtyping	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Project Alpha due
November 16	<b>Study days</b>		
November 18	<b>Semi-finals</b>		
November 20	<b>Semi-finals</b>		
November 23	<b>Semi-finals</b>		
November 25	<b>No class (Thanksgiving)</b>		
November 27	<b>No class (Thanksgiving)</b>		
November 30	Existential Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 2	Propositions as Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 4	Dependent Types	<a href="#">notes</a> <a href="#">video</a> <a href="#">code</a>	Project Beta due
December 7	Concurrency	<a href="#">slides</a> <a href="#">video</a>	Homework 7 due
December 9	Probabilistic Semantics		
December 11	After 4110	<a href="#">slides</a>	
December 14	DSLs and Bidirectional Programming	<a href="#">slides</a> <a href="#">video</a>	Homework 8 due
December 16	Victory Lap		Final Project due

# Semester in Review

CS 4110 Home Project Resources Schedule Syllabus CMS Campuswire

## Schedule

Date	Topic	Notes
September 2	Course Overview	<a href="#">slides</a> <a href="#">video</a>
September 4	Introduction to Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 7	Inductive Definitions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 9	Properties and Inductive Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>
September 11	Inductive Proof and Large-Step Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 14	The IMP Language	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 16	IMP Properties	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 18	More IMP Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">whiteboard-1</a> <a href="#">whiteboard-2</a> <a href="#">video-1</a> <a href="#">video-2</a>
September 21	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 23	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 25	Axiomatic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 28	Hoare Logic	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 30	Hoare Logic Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 2	Weakest Preconditions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 5	$\lambda$ -Calculus	<a href="#">notes</a> <a href="#">video</a>
October 7	More $\lambda$ -Calculus and Substitution	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 9	de Bruijn and Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 12	Encodings	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 14	Mid-semester break	

Mathematical Preliminaries

Operational Semantics

Denotational Semantics

Axiomatic Semantics

$\lambda$ -Calculus

$\lambda$ -Calculus Programming

Simple Types

October 16	Encodings and Fixed-Point Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 19	Definitional Translation	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW4 due
October 21	Continuations	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 23	Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 26	More Types	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	HW5 due
October 28	Proving Type Soundness	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 30	More Proving Type Soundness	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	
November 2	Normalization	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW6 due
November 4	Advanced Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 6	Polymorphism	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 9	Type Inference	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Take-Home Prelim 2
November 11	Recursive Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 13	Records and Subtyping	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Project Alpha due
November 16	<b>Study days</b>		
November 18	<b>Semi-finals</b>		
November 20	<b>Semi-finals</b>		
November 23	<b>Semi-finals</b>		
November 25	<b>No class (Thanksgiving)</b>		
November 27	<b>No class (Thanksgiving)</b>		
November 30	Existential Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 2	Propositions as Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 4	Dependent Types	<a href="#">notes</a> <a href="#">video</a> <a href="#">code</a>	Project Beta due
December 7	Concurrency	<a href="#">slides</a> <a href="#">video</a>	Homework 7 due
December 9	Probabilistic Semantics		
December 11	After 4110	<a href="#">slides</a>	
December 14	DSLs and Bidirectional Programming	<a href="#">slides</a> <a href="#">video</a>	Homework 8 due
December 16	Victory Lap		Final Project due

# Semester in Review

CS 4110 Home Project Resources Schedule Syllabus CMS Campuswire

## Schedule

Date	Topic	Notes
September 2	Course Overview	<a href="#">slides</a> <a href="#">video</a>
September 4	Introduction to Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 7	Inductive Definitions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 9	Properties and Inductive Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>
September 11	Inductive Proof and Large-Step Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 14	The IMP Language	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 16	IMP Properties	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 18	More IMP Proofs	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">whiteboard-1</a> <a href="#">whiteboard-2</a> <a href="#">video-1</a> <a href="#">video-2</a>
September 21	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 23	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 25	Axiomatic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 28	Hoare Logic	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 30	Hoare Logic Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 2	Weakest Preconditions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 5	$\lambda$ -Calculus	<a href="#">notes</a> <a href="#">video</a>
October 7	More $\lambda$ -Calculus and Substitution	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 9	de Bruijn and Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 12	Encodings	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 14	Mid-semester break	

Mathematical Preliminaries

Operational Semantics

Denotational Semantics

Axiomatic Semantics

$\lambda$ -Calculus

$\lambda$ -Calculus Programming

Simple Types

Advanced Types

October 16	Encodings and Fixed-Point Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 19	Definitional Translation	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW4 due
October 21	Continuations	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 23	Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 26	More Types	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	HW5 due
October 28	Proving Type Soundness	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 30	More Proving Type Soundness	<a href="#">same slides</a> <a href="#">same notes</a> <a href="#">video</a>	
November 2	Normalization	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW6 due
November 4	Advanced Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 6	Polymorphism	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 9	Type Inference	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Take-Home Prelim 2
November 11	Recursive Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 13	Records and Subtyping	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Project Alpha due
November 16	Study days		
November 18	Semi-finals		
November 20	Semi-finals		
November 23	Semi-finals		
November 25	No class (Thanksgiving)		
November 27	No class (Thanksgiving)		
November 30	Existential Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 2	Propositions as Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 4	Dependent Types	<a href="#">notes</a> <a href="#">video</a> <a href="#">code</a>	Project Beta due
December 7	Concurrency	<a href="#">slides</a> <a href="#">video</a>	Homework 7 due
December 9	Probabilistic Semantics		
December 11	After 4110	<a href="#">slides</a>	
December 14	DSLs and Bidirectional Programming	<a href="#">slides</a> <a href="#">video</a>	Homework 8 due
December 16	Victory Lap		Final Project due

# Semester in Review

CS 4110 Home Project Resources Schedule Syllabus CMS Campuswire

## Schedule

Date	Topic	Notes
September 2	Course Overview	<a href="#">slides</a> <a href="#">video</a>
September 4	Introduction to Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 7	Inductive Definitions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 9	Properties and Inductive Proofs	same slides same notes <a href="#">video</a>
September 11	Inductive Proof and Large-Step Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 14	The IMP Language	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 16	IMP Properties	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 18	More IMP Proofs	same slides same notes <a href="#">whiteboard-1</a> <a href="#">whiteboard-2</a> <a href="#">video-1</a> <a href="#">video-2</a>
September 21	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 23	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 25	Axiomatic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 28	Hoare Logic	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 30	Hoare Logic Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 2	Weakest Preconditions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 5	$\lambda$ -Calculus	<a href="#">notes</a> <a href="#">video</a>
October 7	More $\lambda$ -Calculus and Substitution	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 9	de Bruijn and Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 12	Encodings	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 14	Mid-semester break	

Mathematical Preliminaries

Operational Semantics

Denotational Semantics

Axiomatic Semantics

$\lambda$ -Calculus

$\lambda$ -Calculus Programming

Simple Types

Advanced Types

Logic and Proof

October 16	Encodings and Fixed-Point Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 19	Definitional Translation	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW4 due
October 21	Continuations	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 23	Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 26	More Types	same slides same notes <a href="#">video</a>	HW5 due
October 28	Proving Type Soundness	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
October 30	More Proving Type Soundness	same slides same notes <a href="#">video</a>	
November 2	Normalization	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	HW6 due
November 4	Advanced Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 6	Polymorphism	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 9	Type Inference	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Take-Home Prelim 2
November 11	Recursive Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
November 13	Records and Subtyping	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Project Alpha due
November 16	<b>Study days</b>		
November 18	<b>Semi-finals</b>		
November 20	<b>Semi-finals</b>		
November 23	<b>Semi-finals</b>		
November 25	<b>No class (Thanksgiving)</b>		
November 27	<b>No class (Thanksgiving)</b>		
November 30	Existential Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 2	Propositions as Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	
December 4	Dependent Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Project Beta due
December 7	Concurrency	<a href="#">slides</a> <a href="#">video</a>	Homework 7 due
December 9	Probabilistic Semantics		
December 11	After 4110	<a href="#">slides</a>	
December 14	DSLs and Bidirectional Programming	<a href="#">slides</a> <a href="#">video</a>	Homework 8 due
December 16	Victory Lap		Final Project due

# Semester in Review

CS 4110 Home Project Resources Schedule Syllabus CMS Campuswire

## Schedule

Date	Topic	Notes
September 2	Course Overview	<a href="#">slides</a> <a href="#">video</a>
September 4	Introduction to Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 7	Inductive Definitions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 9	Properties and Inductive Proofs	same slides same notes <a href="#">video</a>
September 11	Inductive Proof and Large-Step Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 14	The IMP Language	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 16	IMP Properties	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 18	More IMP Proofs	same slides same notes <a href="#">whiteboard-1</a> <a href="#">whiteboard-2</a> <a href="#">video-1</a> <a href="#">video-2</a>
September 21	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 23	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 25	Axiomatic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 28	Hoare Logic	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 30	Hoare Logic Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 2	Weakest Preconditions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 5	$\lambda$ -Calculus	<a href="#">notes</a> <a href="#">video</a>
October 7	More $\lambda$ -Calculus and Substitution	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 9	de Bruijn and Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 12	Encodings	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 14	Mid-semester break	

Mathematical Preliminaries

Operational Semantics

Denotational Semantics

Axiomatic Semantics

$\lambda$ -Calculus

October 16	Encodings and Fixed-Point Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	$\lambda$ -Calculus Programming	HW4 due
October 19	Definitional Translation	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 21	Continuations	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 23	Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 26	More Types	same slides same notes <a href="#">video</a>	Simple Types	HW5 due
October 28	Proving Type Soundness	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 30	More Proving Type Soundness	same slides same notes <a href="#">video</a>		
November 2	Normalization	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		HW6 due
November 4	Advanced Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
November 6	Polymorphism	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Advanced Types	
November 9	Type Inference	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		Take-Home Prelim 2
November 11	Recursive Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
November 13	Records and Subtyping	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		Project Alpha due
November 16	Study days			
November 18	Semi-finals			
November 20	Semi-finals			
November 23	Semi-finals			
November 25	No class (Thanksgiving)			
November 27	No class (Thanksgiving)			
November 30	Existential Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
December 2	Propositions as Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Logic and Proof	Project Beta due
December 4	Dependent Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
December 7	Concurrency	<a href="#">slides</a> <a href="#">video</a>		Homework 7 due
December 9	Probabilistic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Advanced Topics	
December 11	After 4110	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
December 14	DSLs and Bidirectional Programming	<a href="#">slides</a> <a href="#">video</a>		Homework 8 due
December 16	Victory Lap			Final Project due

# Semester in Review

CS 4110 Home Project Resources Schedule Syllabus CMS Campuswire

## Schedule

Date	Topic	Notes
September 2	Course Overview	<a href="#">slides</a> <a href="#">video</a>
September 4	Introduction to Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 7	Inductive Definitions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 9	Properties and Inductive Proofs	same slides same notes <a href="#">video</a>
September 11	Inductive Proof and Large-Step Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 14	The IMP Language	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 16	IMP Properties	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 18	More IMP Proofs	same slides same notes <a href="#">whiteboard-1</a> <a href="#">whiteboard-2</a> <a href="#">video-1</a> <a href="#">video-2</a>
September 21	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 23	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 25	Axiomatic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 28	Hoare Logic	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 30	Hoare Logic Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 2	Weakest Preconditions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 5	$\lambda$ -Calculus	<a href="#">notes</a> <a href="#">video</a>
October 7	More $\lambda$ -Calculus and Substitution	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 9	de Bruijn and Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 12	Encodings	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 14	Mid-semester break	

Mathematical Preliminaries

Operational Semantics

Denotational Semantics

Axiomatic Semantics

$\lambda$ -Calculus

Proofs

October 16	Encodings and Fixed-Point Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	$\lambda$ -Calculus Programming	HW4 due
October 19	Definitional Translation	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 21	Continuations	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 23	Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 26	More Types	same slides same notes <a href="#">video</a>	Simple Types	HW5 due
October 28	Proving Type Soundness	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 30	More Proving Type Soundness	same slides same notes <a href="#">video</a>		
November 2	Normalization	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		HW6 due
November 4	Advanced Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
November 6	Polymorphism	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Advanced Types	
November 9	Type Inference	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		Take-Home Prelim 2
November 11	Recursive Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
November 13	Records and Subtyping	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		Project Alpha due
November 16	Study days			
November 18	Semi-finals			
November 20	Semi-finals			
November 23	Semi-finals			
November 25	No class (Thanksgiving)			
November 27	No class (Thanksgiving)			
November 30	Existential Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
December 2	Propositions as Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Logic and Proof	Project Beta due
December 4	Dependent Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
December 7	Concurrency	<a href="#">slides</a> <a href="#">video</a>		Homework 7 due
December 9	Probabilistic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Advanced Topics	
December 11	After 4110			
December 14	DSLs and Bidirectional Programming	<a href="#">slides</a> <a href="#">video</a>		Homework 8 due
December 16	Victory Lap			Final Project due

$\lambda$ -Calculus Programming

Simple Types

Advanced Types

Logic and Proof

Advanced Topics

# Semester in Review

CS 4110 Home Project Resources Schedule Syllabus CMS Campuswire

## Schedule

Date	Topic	Notes
September 2	Course Overview	<a href="#">slides</a> <a href="#">video</a>
September 4	Introduction to Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 7	Inductive Definitions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 9	Properties and Inductive Proofs	same slides same notes video
September 11	Inductive Proof and Large-Step Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 14	The IMP Language	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 16	IMP Properties	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 18	More IMP Proofs	same slides same notes whiteboard-1 whiteboard-2 video-1 video-2
September 21	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 23	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 25	Axiomatic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 28	Hoare Logic	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 30	Hoare Logic Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 2	Weakest Preconditions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 5	$\lambda$ -Calculus	<a href="#">notes</a> <a href="#">video</a>
October 7	More $\lambda$ -Calculus and Substitution	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 9	de Bruijn and Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 12	Encodings	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 14	Mid-semester break	

Mathematical Preliminaries

Operational Semantics

Denotational Semantics

Axiomatic Semantics

$\lambda$ -Calculus

October 16	Encodings and Fixed-Point Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	$\lambda$ -Calculus Programming	HW4 due
October 19	Definitional Translation	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 21	Continuations	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 23	Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 26	More Types	same slides same notes video	Simple Types	HW5 due
October 28	Proving Type Soundness	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 30	More Proving Type Soundness	same slides same notes video		
November 2	Normalization	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		HW6 due
November 4	Advanced Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
November 6	Polymorphism	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Advanced Types	
November 9	Type Inference	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		Take-Home Prelim 2
November 11	Recursive Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
November 13	Records and Subtyping	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		Project Alpha due
November 16	Study days			
November 18	Semi-finals			
November 20	Semi-finals			
November 23	Semi-finals			
November 25	No class (Thanksgiving)			
November 27	No class (Thanksgiving)			
November 30	Existential Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
December 2	Propositions as Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Logic and Proof	Project Beta due
December 4	Dependent Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
December 7	Concurrency	<a href="#">slides</a> <a href="#">video</a>		Homework 7 due
December 9	Probabilistic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Advanced Topics	
December 11	After 4110			
December 14	DSLs and Bidirectional Programming	<a href="#">slides</a> <a href="#">video</a>		Homework 8 due
December 16	Victory Lap			Final Project due

Proofs

Interpreters

# Semester in Review

CS 4110 Home Project Resources Schedule Syllabus CMS Campuswire

## Schedule

Date	Topic	Notes
September 2	Course Overview	<a href="#">slides</a> <a href="#">video</a>
September 4	Introduction to Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 7	Inductive Definitions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 9	Properties and Inductive Proofs	same slides same notes video
September 11	Inductive Proof and Large-Step Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 14	The IMP Language	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 16	IMP Properties	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 18	More IMP Proofs	same slides same notes whiteboard-1 whiteboard-2 video-1 video-2
September 21	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 23	Denotational Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 25	Axiomatic Semantics	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 28	Hoare Logic	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
September 30	Hoare Logic Examples	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 2	Weakest Preconditions	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 5	$\lambda$ -Calculus	<a href="#">notes</a> <a href="#">video</a>
October 7	More $\lambda$ -Calculus and Substitution	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 9	de Bruijn and Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 12	Encodings	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>
October 14	Mid-semester break	

Mathematical Preliminaries

Operational Semantics

Denotational Semantics

Axiomatic Semantics

$\lambda$ -Calculus

October 16	Encodings and Fixed-Point Combinators	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	$\lambda$ -Calculus Programming	HW4 due
October 19	Definitional Translation	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 21	Continuations	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 23	Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 26	More Types	same slides same notes video	Simple Types	HW5 due
October 28	Proving Type Soundness	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
October 30	More Proving Type Soundness	same slides same notes video		
November 2	Normalization	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		HW6 due
November 4	Advanced Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
November 6	Polymorphism	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Advanced Types	
November 9	Type Inference	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		Take-Home Prelim 2
November 11	Recursive Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
November 13	Records and Subtyping	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		Project Alpha due
November 16	Study days			
November 18	Semi-finals			
November 20	Semi-finals			
November 23	Semi-finals			
November 25	No class (Thanksgiving)			
November 27	No class (Thanksgiving)			
November 30	Existential Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
December 2	Propositions as Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>	Logic and Proof	Project Beta due
December 4	Dependent Types	<a href="#">slides</a> <a href="#">notes</a> <a href="#">video</a>		
December 7	Concurrency	<a href="#">slides</a> <a href="#">video</a>		Homework 7 due
December 9	Probabilistic Semantics		Advanced Topics	
December 11	After 4110			
December 14	DSLs and Bidirectional Programming	<a href="#">slides</a> <a href="#">video</a>		Homework 8 due
December 16	Victory Lap			Final Project due

Proofs

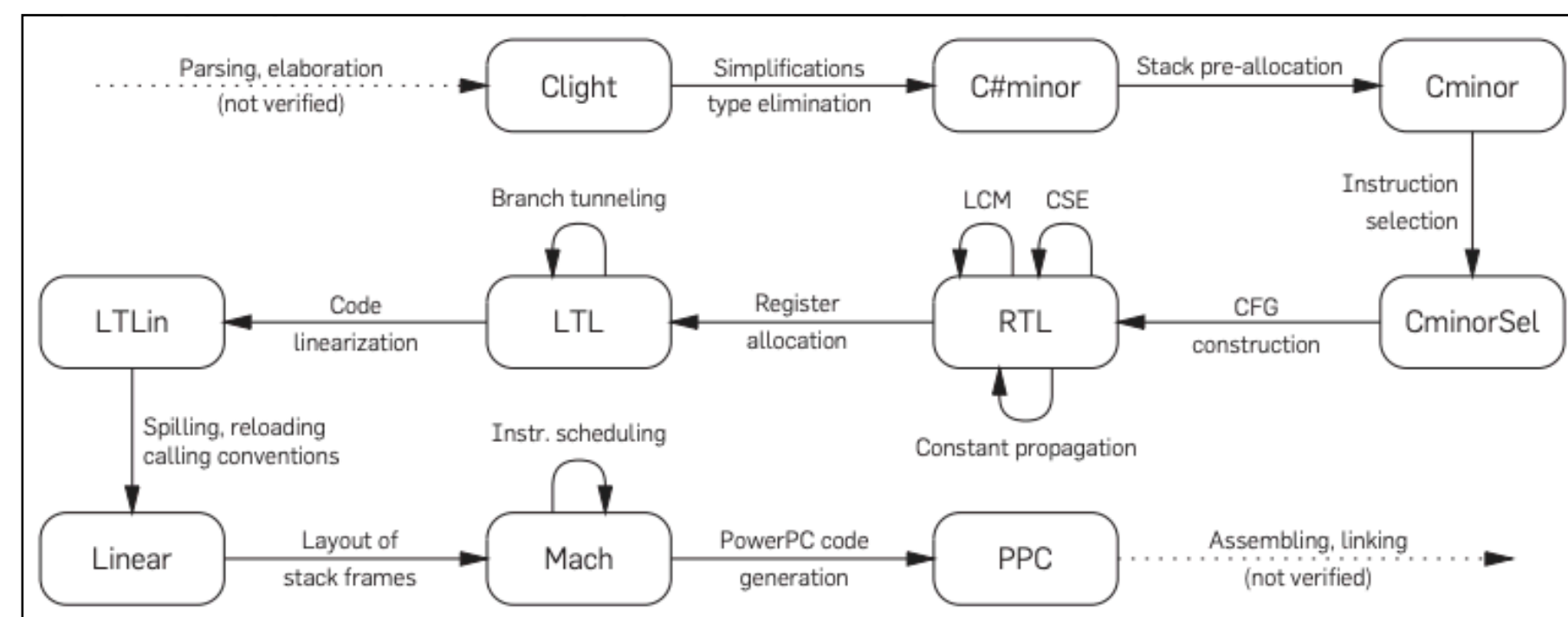
Interpreters

Verifiers



# Trend: Verified Software

- Researchers are increasingly developing operational models of real-world languages
- Progress in verification makes it possible to build end-to-end verified systems
- Example: CompCert
  - Formal semantics for (a large subset of C)
  - Formal semantics for PPC (and later x86)
  - Full compiler verified in Coq



DOI:10.1145/1538788.1538814

## Formal Verification of a Realistic Compiler

By Xavier Leroy

### Abstract

This paper reports on the development and formal verification (proof of semantic preservation) of CompCert, a compiler from Clight (a large subset of the C programming language) to PowerPC assembly code, using the Coq proof assistant both for programming the compiler and for proving its correctness. Such a verified compiler is useful in the context of critical software and its formal verification: the verification of the compiler guarantees that the safety properties proved on the source code hold for the executable compiled code as well.

### 1. INTRODUCTION

Can you trust your compiler? Compilers are generally assumed to be semantically transparent: the compiled code should behave as prescribed by the semantics of the source program. Yet, compilers—and especially optimizing compilers—are complex programs that perform complicated symbolic transformations. Despite intensive testing, bugs in compilers do occur, causing the compilers to crash at compile-time or—much worse—to silently generate an incorrect executable for a correct source program.

For low-assurance software, validated only by testing, the impact of compiler bugs is low: what is tested is the executable code produced by the compiler; rigorous testing should expose compiler-introduced errors along with errors already present in the source program. Note, however, that compiler-introduced bugs are notoriously difficult to expose and track down. The picture changes dramatically for safety-critical, high-assurance software. Here, validation by testing reaches its limits and needs to be complemented or replaced by the use of formal methods and models

preserves the semantics of the source programs. For the last 5 years, we have been working on the development of a *realistic, verified* compiler called CompCert. By *verified*, we mean a compiler that is accompanied by a machine-checked proof of a semantic preservation property: the generated machine code behaves as prescribed by the semantics of the source program. By *realistic*, we mean a compiler that could realistically be used in the context of production of critical software. Namely, it compiles a language commonly used for critical embedded software: neither Java nor ML nor assembly code, but a large subset of the C language. It produces code for a processor commonly used in embedded systems: we chose the PowerPC because it is popular in avionics. Finally, the compiler must generate code that is efficient enough and compact enough to fit the requirements of critical embedded systems. This implies a multipass compiler that features good register allocation and some basic optimizations.

Proving the correctness of a compiler is by no ways a new idea: the first such proof was published in 1967<sup>16</sup> (for the compilation of arithmetic expressions down to stack machine code) and mechanically verified in 1972.<sup>17</sup> Since then, many other proofs have been conducted, ranging from single-pass compilers for toy languages to sophisticated code optimizations.<sup>8</sup> In the CompCert experiment, we carry this line of work all the way to end-to-end verification of a complete compilation chain from a structured imperative language down to assembly code through eight intermediate languages. While conducting the verification of CompCert, we found that many of the nonoptimizing translations performed, while often considered obvious in the compiler literature, are surprisingly tricky to formally prove correct.

This paper gives a high-level overview of the CompCert compiler and its mechanical verification, which is the

## Petr4: Formal Foundations for P4 Data Planes

RYAN DOENGES, Cornell University, USA  
MINA TAHMASBI ARASHLOO, Cornell University, USA  
ALEXANDER CHANG, Cornell University, USA  
NEWTON NI, Cornell University, USA  
SAMWISE PARKINSON, Cornell University, USA  
RUDY PETERSON, Cornell University, USA  
ALAIA SOLKO-BRESLIN, Cornell University, USA  
AMANDA XU, Cornell University, USA  
NATE FOSTER, Cornell University, USA



P4 is a domain-specific language for specifying the behavior of packet-processing systems. It is based on an elegant design with high-level abstractions, such as parsers and match-action pipelines, which can be compiled to efficient implementations in hardware or software. Unfortunately, like many industrial languages, P4 lacks a formal foundation. The P4 specification is a 160-page document with a mixture of informal prose, graphical diagrams, and pseudocode. The reference compiler is complex, running to over 40KLoC of C++ code. Clearly neither of these artifacts is suitable for formal reasoning.

This paper presents a new framework, called PETR4, that puts P4 on a solid foundation. PETR4 uses standard elements of the semantics engineering toolkit, namely type systems and operational semantics, to build a compositional semantics that assigns an unambiguous meaning to every P4 program. PETR4 is implemented as an OCaml prototype that has been validated against a suite of over 750 tests from the reference implementation. While developing PETR4, we discovered dozens of bugs in the language specification and the reference implementation, many of which have been fixed. Furthermore, we have used PETR4 to establish the soundness of P4's type system, prove key properties such as termination, and formalize a language extension.

# The Petr4 Team



Ryan Doenges



Mina Tahmasbi Arashloo



Santiago Bautista



Alexander Chang



Newton Ni



Samwise Parkinson



Rudy Peterson



Alaia Solko-Breslin



Amanda Xu



Nate Foster

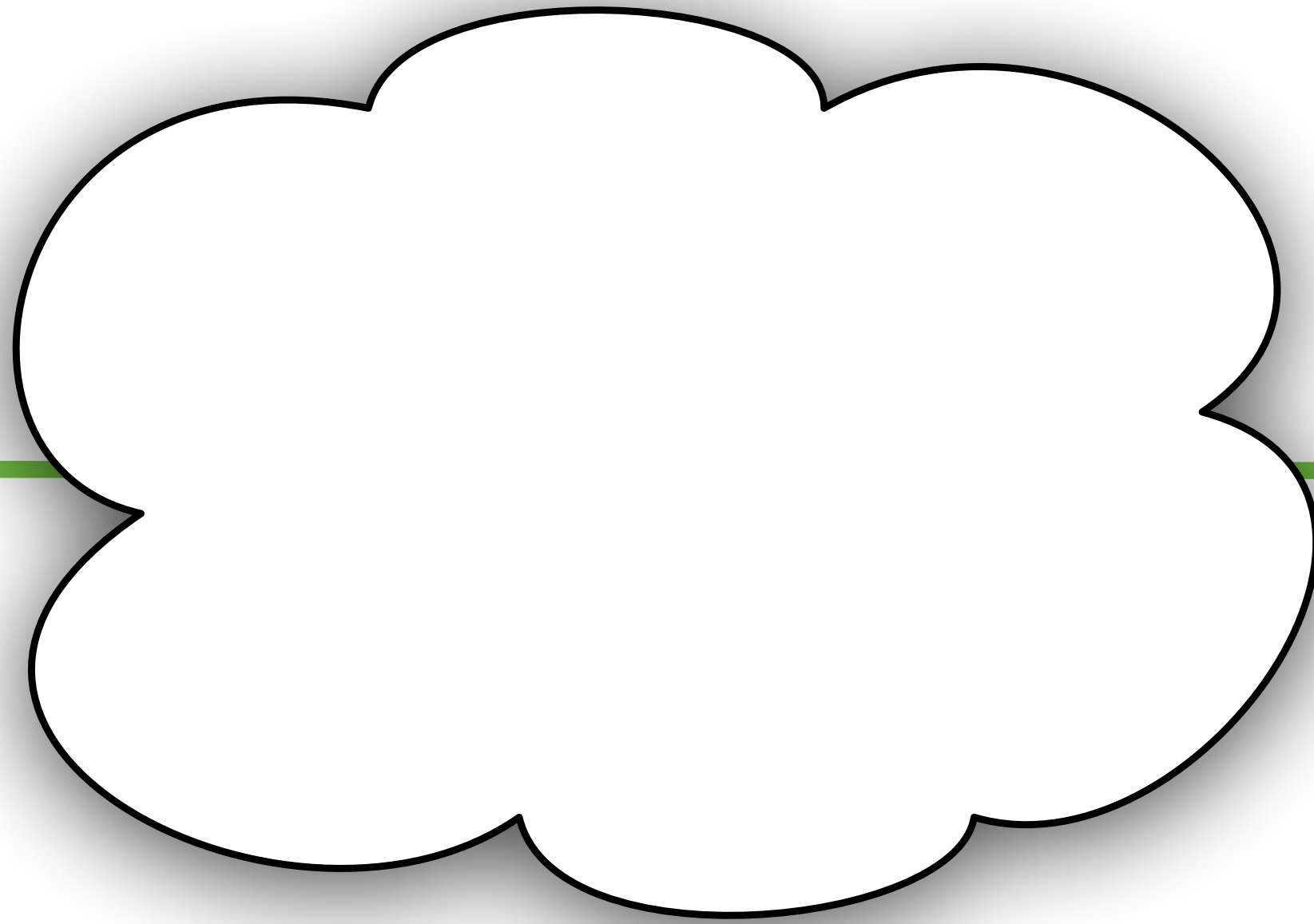


Let's see what's going on in  
the Tour this morning!



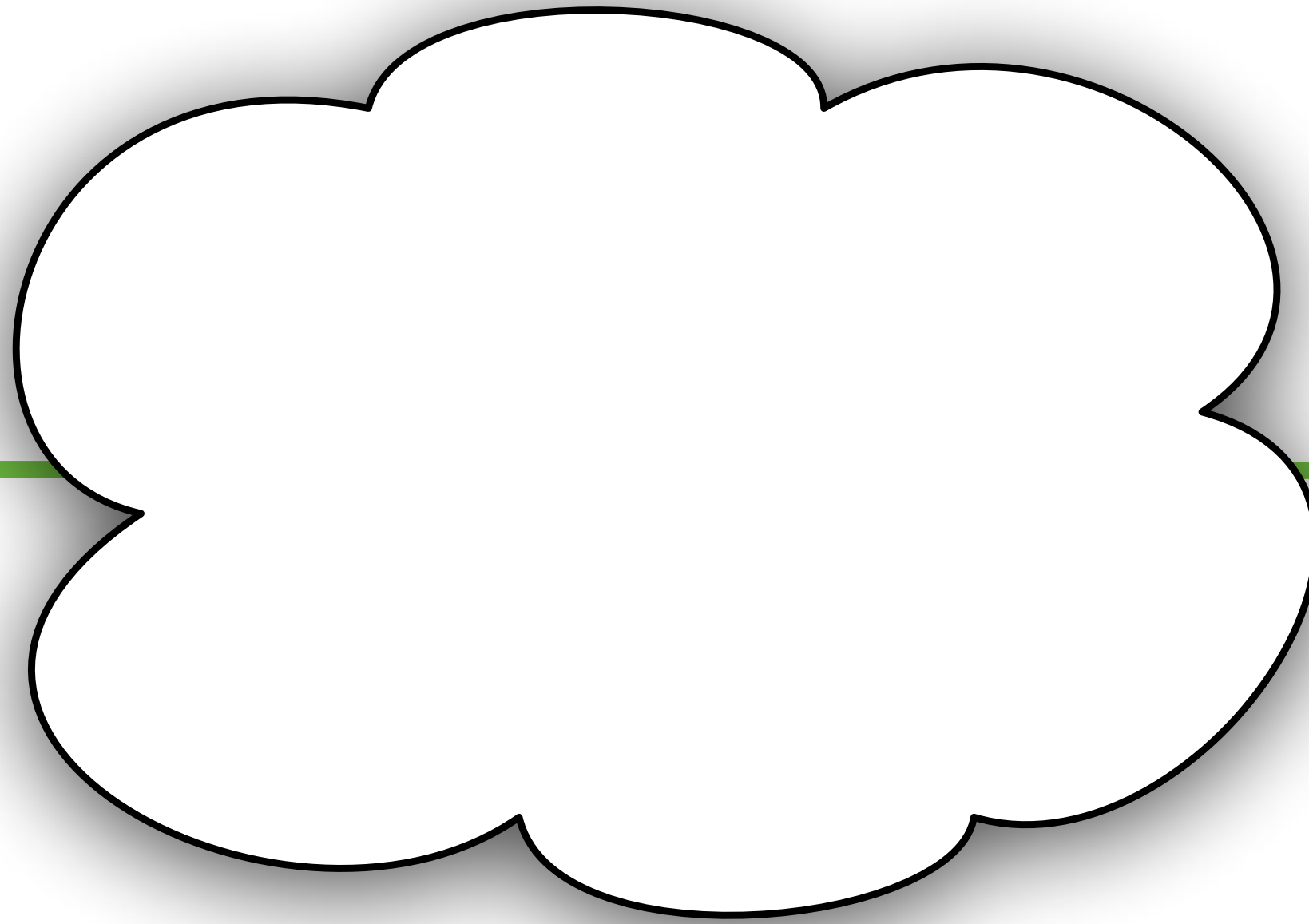


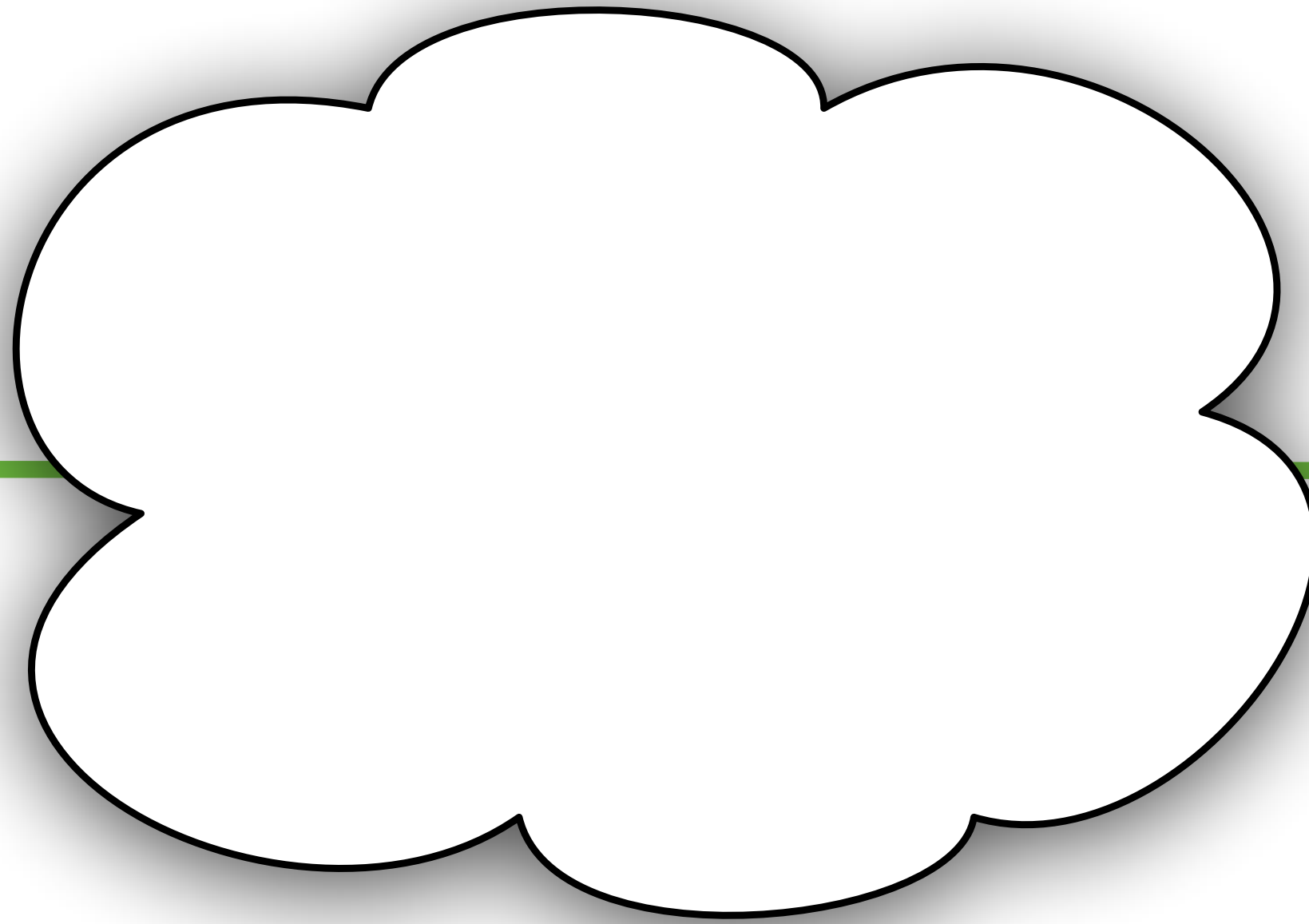


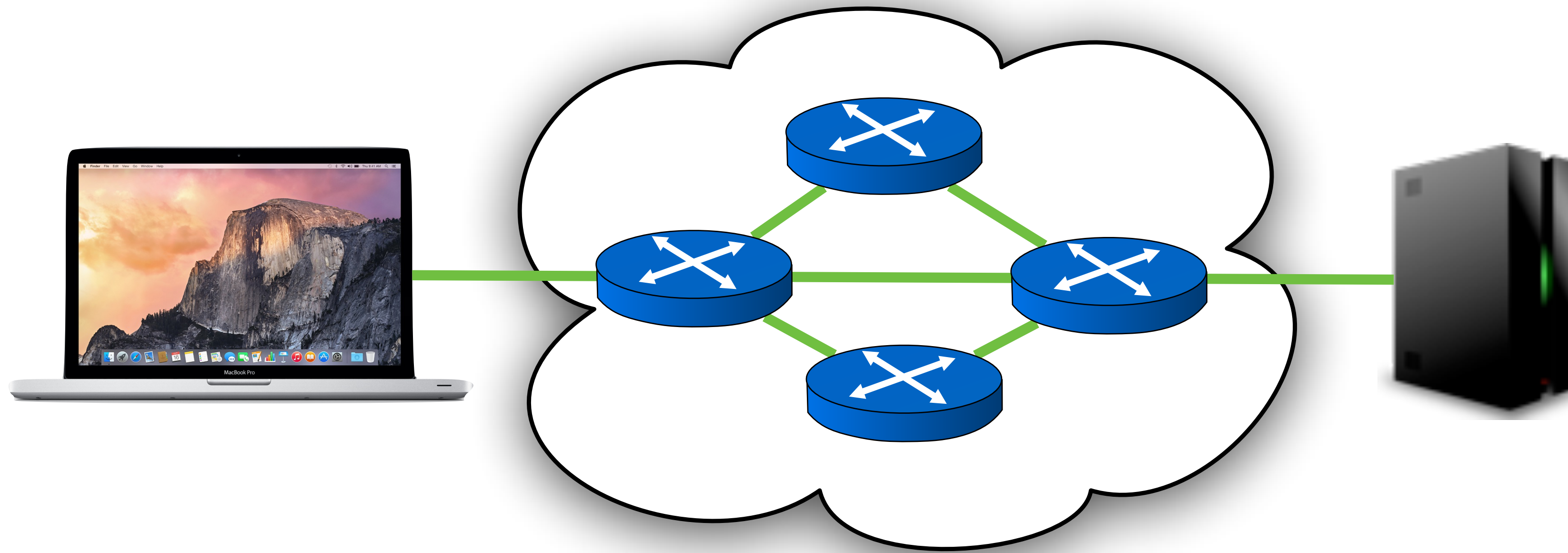




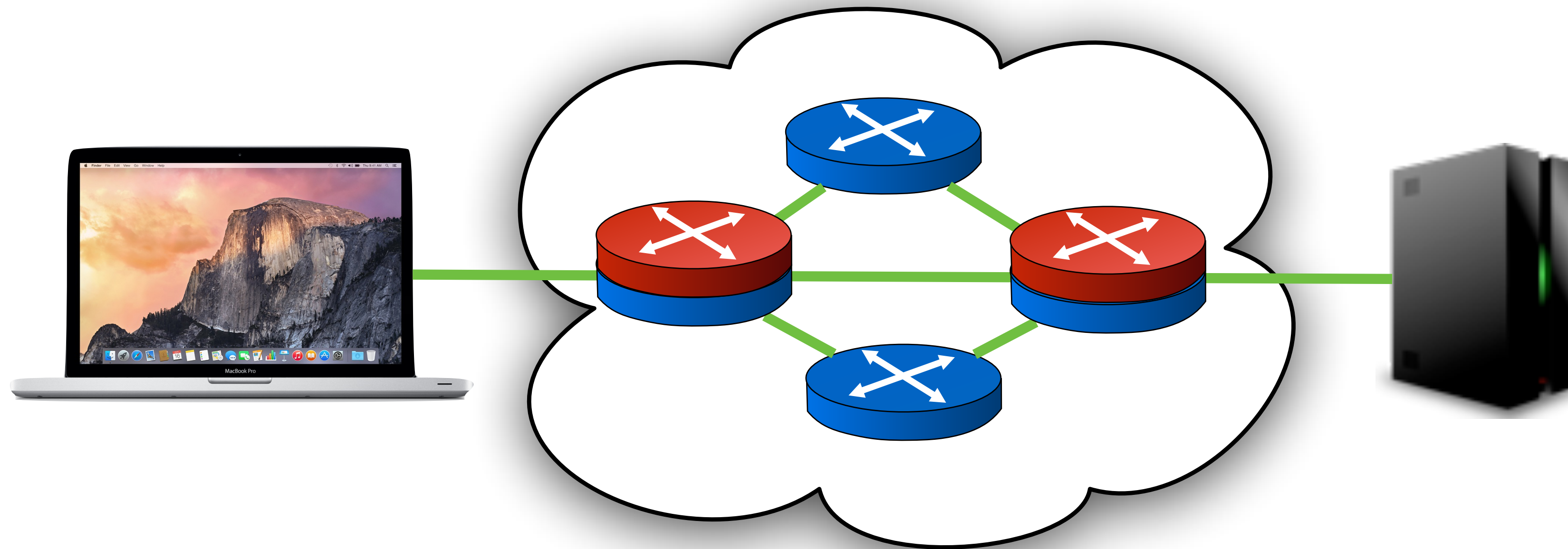




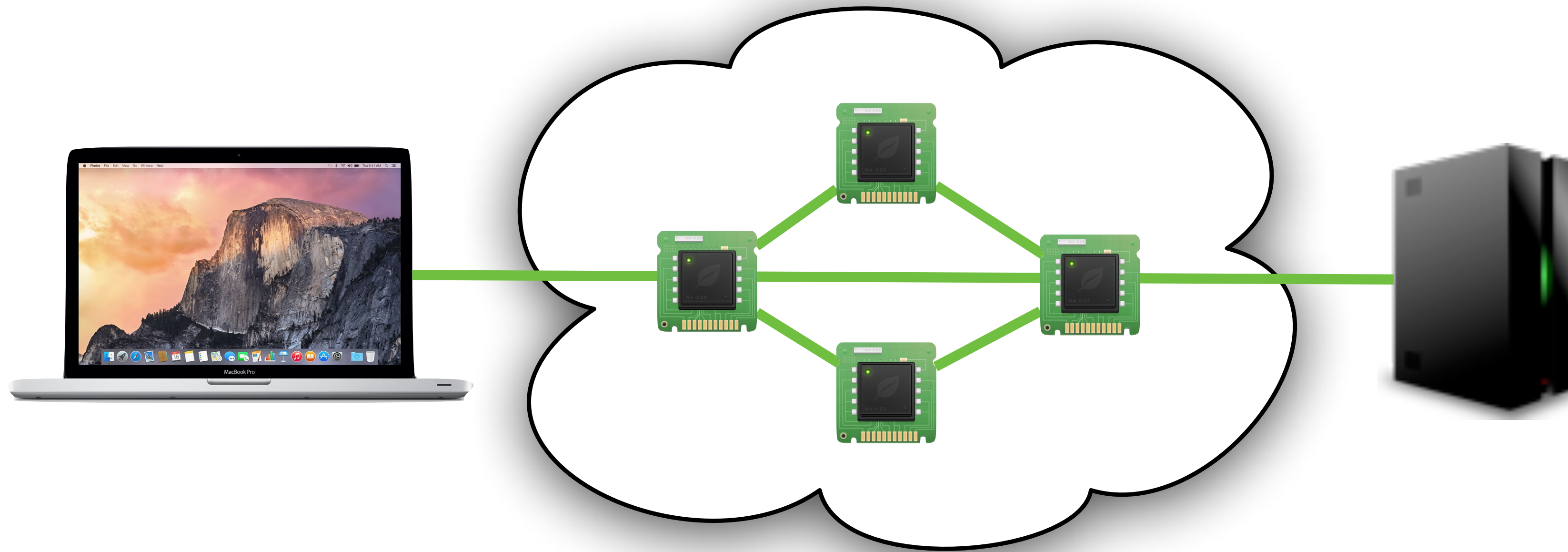




**Data plane:** forward packets, balances load, implements monitoring, etc.

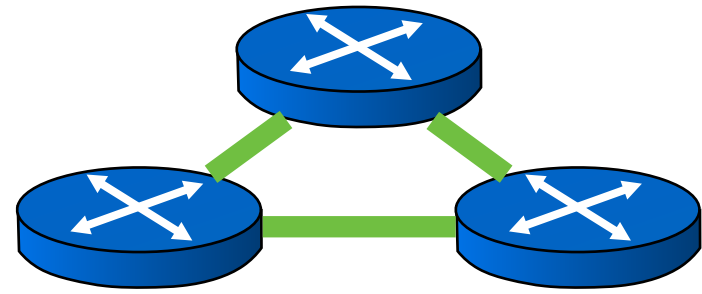


**Control Plane:** discovers topology, computes routes, enforces policies, etc.

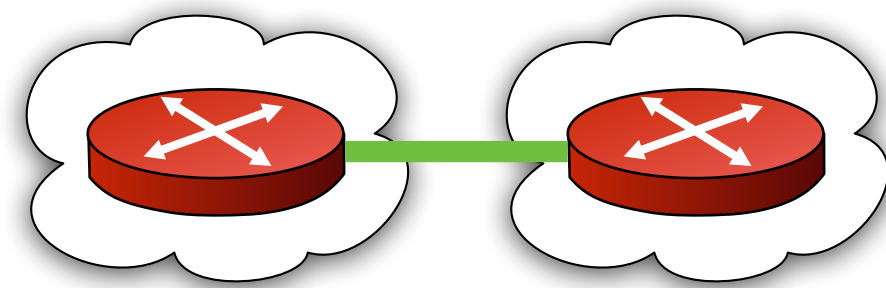


**Network Devices:** implement packet processing, buffering, queueing, etc. at line rate

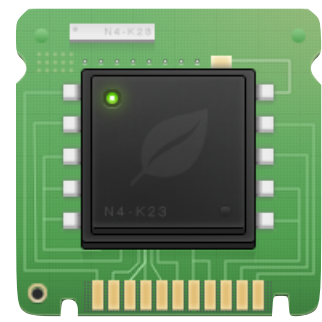
# Network Programming Challenges



Networks are *distributed* systems with thousands of interacting nodes



Networks enforce complex *security* policies that span trust boundaries



Networks are expected to offer good *performance* with limited resources

# Brief History of Network Verification

## On Static Reachability Analysis of IP Networks

Geoffrey G. Xie<sup>1</sup> Jibin Zhan<sup>2</sup> David A. Maltz<sup>2</sup> Hui Zhang<sup>1</sup>  
 Albert Greenberg<sup>1</sup> Giali Hjalmtsson<sup>3</sup> Jennifer Rexford<sup>1</sup>

### ABSTRACT

The primary purpose of a network is to provide reachability between applications running on end hosts. In this paper, we describe how to compute the reachability of a network from a snapshot of the configuration state from each of the routers. Our primary contribution is the precise definition of the *potential reachability* of a network and a substantial simplification of the problem through a unified modeling of packet filters and routing protocols. In the end, we reduce a complex, important practical problem to computing the transitive closure to set union and intersection operations on reachability set representations. We then extend our algorithm to model the influence of packet transformations (e.g., by NAT or ToS remapping) along the path. Our technique for static analysis of network reachability is valuable for verifying the intent of the network designer, troubleshooting reachability problems, and performing “what-if” analysis of failure scenarios.

**Index Terms**—Routing, Static Configuration Analysis.

### 1. INTRODUCTION

While the ultimate goal of networking is to enable communication between hosts that are not directly connected, a wide variety of mechanisms are being used to *limit* the set of destinations the hosts can reach. For example, backbone networks may provide Virtual Private Network services to connect only remote offices belonging to the same enterprise, and enterprise networks themselves are often segmented into departments or offices whose hosts must be isolated for business or security reasons. Also, due to configuration or design mistakes, two hosts may not be able to communicate under certain failure scenarios, even though the network remains connected, knowing when these vulnerabilities exist is crucial for building a more reliable network.

Research sponsored by the NSF under ANI-0085920, ANI-0318653, and ANI-040164. Views and conclusions contained in this document are those of the authors.

<sup>1</sup>Northeastern University, {gixie@psd.edu, hzhang@ccs.nyu.edu, ggreen@ccs.nyu.edu}

<sup>2</sup>Carnegie Mellon University, {jibin.zhan@cmu.edu, dmaltz@cmu.edu}

<sup>3</sup>AT&T Labs-Research, {giali.hjalmtsson@att.com}

Determining what kinds of packets can be exchanged between two hosts connected to a network is a difficult and critical problem facing network designers and operators. To our knowledge, the problem is largely unexamined in the networking research literature. Solving the problem requires knowing far more than the network’s topology or the routing protocols it uses. For example, despite having a route to a remote end-point, a sender’s packets may be discarded by a packet filter on one of the links in the path. The network’s packet filters, routing policies, and packet transformations all must be taken into account to even ask the simple and very important question of “can those two hosts communicate?”

This paper crystallizes the problem of calculating the reachability provided by a network. By mapping packet filters, routing information, and packet transformations to a single unified model of reachability we have determined how to transform this seemingly intractable problem into a classical graph problem that can be solved with polynomial time algorithms such as transitive closure. This is the primary contribution of this paper.

### A. Advantages of Automated Static Analysis

Currently, the common practice to determine if packets can reach from one point in a network to another is to use tools such as `ping` and `traceroute` to send probe traffic that experimentally test whether reachability exists. In contrast, we have developed a *static-analysis* approach that can be applied even if only a description of the network is available. Static analysis has many advantages over ping and `traceroute`, including:

- The ability to determine a description of the set of packets that *could* traverse the network from a given starting point to a given ending point, whereas experimental techniques can only check the reachability of the specific probe traffic they send.
- The ability to calculate the set of routers and hosts that a given packet could potentially reach, whereas `ping` and `traceroute` can only check reachability along the path currently selected by the routing protocols.
- The ability to evaluate the reachability of a network during its *design phase*—before the network

## Header Space Analysis: Static Checking For Networks

Peyman Kazemian George Varghese Nick McKeown  
 Stanford University UCSD and Yahoo! Research Stanford University  
 kazemian@stanford.edu varghese@cs.ucsd.edu nickm@stanford.edu

### ABSTRACT

Today’s networks typically carry or deploy dozens of protocols and mechanisms simultaneously such as MPLS, NAT, ACLs and route redistribution. Even when individual protocols function correctly, failures can arise from the complex interactions of their aggregate, requiring network administrators to be masters of detail. Our goal is to automatically find an important class of failures, regardless of the protocols running, for both operational and experimental networks.

To this end we developed a general and protocol-agnostic framework, called *Header Space Analysis* (HSA). Our formalism allows us to statically check network specifications and configurations to identify an important class of failures such as *Reachability Failures*, *Forwarding Loops* and *Traffic Isolation and Leakage* problems. In HSA, protocol header fields are not first class entities; instead we look at the entire packet header as a concatenation of bits without any associated meaning. Each packet is a point in the  $\{0,1\}^L$  space where  $L$  is the maximum length of a packet header, and networking boxes transform packets from one point in the space to another point or set of points (multisets).

We created a library of tools, called *HSAset*, to implement our framework, and used it to analyze a variety of networks and protocols. *HSAset* was used to analyze the Stanford University backbone network, and found all the forwarding loops in less than 10 minutes, and verified reachability constraints between two hosts in 13 seconds. It also found a large and complex loop in an experimental loose source routing protocol in 4 minutes.

### 1. Introduction

“Accidents will occur in the best-regulated families” — Charles Dickens

In the beginning, a switch or router was breathtakingly simple. About all the device needed to do was index into a forwarding table using a destination address, and decide where to send the packet next. Over time, forwarding grew more complicated. Middleboxes (e.g., NAT and firewalls) and encapsulation mechanisms (e.g., VLAN and MPLS) appeared to escape from IP’s limitations: e.g., NAT bypasses address limits and MPLS

allows flexible routing. Further, new protocols for specific domains, such as data centers, WANs and wireless, have greatly increased the complexity of packet forwarding. Today, there are over 6,000 Internet RFCs and it is not unusual for a switch or router to handle ten or more encapsulation format simultaneous.

This complexity makes it daunting to operate a large network today. Network operators require great sophistication to master the complexity of many interacting protocols and middleboxes. The future is not any more trying-complexity today makes operators wary of trying new protocols, even if they are available, for fear of breaking their network. Complexity also makes networks fragile, and susceptible to problems where hosts become isolated and unable to communicate. Debugging reachability problems is very time consuming. Even simple questions are hard to answer, such as “Can Host A talk to Host B?” or “Can packets loop in my network?” or “Can User A listen to communications between User B and C?” These questions are especially hard to answer in networks carrying multiple encapsulations and containing boxes that filter packets.

Thus, our first goal is to help system administrators statically analyze production networks today. We describe new methods and tools to provide formal answers to these questions, and many other failure conditions, regardless of the protocols running in the network. Our second goal is to make it easier for system administrators to guarantee isolation between sets of hosts, users or traffic. Partitioning networks this way is usually called “slicing.” VLANs are a simple example used today. If configured correctly, we can be confident that traffic in one slice (e.g. a VLAN) cannot leak into another. This is useful for security, and to help answer questions such as “Can I prevent Host A from talking to Host B?”. For example, imagine two health-care providers using the same physical network. HIPAA [20] rules require that no information about a patient can be read by other providers. Thus a natural application of slicing is to place each provider in a separate slice and guarantee that no packet from one slice can be controlled by or read by the other slice. We call this *secure slicing*. Secure slicing may also be useful for banks as part of defense-in-depth, and for classified and unclassified users sharing the same physical network. Our tools can verify that slices have

## Network-Wide Invariants in Real Time

Xuan Zhou, Matthew Casar, P. Brighten Godfrey  
 Illinois at Urbana-Champaign  
 {xzhou10, casar, pbg}@illinois.edu

complexity of software will increase. Moreover, SDN allows multiple applications or even multiple users to program the same physical network simultaneously, potentially resulting in conflicting rules that alter the intended behavior of one or more applications [25].

One solution is to rigorously check network software or configuration for bugs prior to deployment. Symbolic execution [12] can catch bugs through exploration of all possible code paths, but is usually not tractable for large software. Analysis of configuration files [13, 28] is useful, but cannot find bugs in router software, and must be designed for specific configuration languages and control protocols. Moreover, using these approaches, an operator who wants to ensure the network’s correctness must have access to the software and configuration, which may be inconvenient in an SDN network where controllers can be operated by other parties [25]. Another approach is to statically analyze snapshots of the network-wide data-plane state [9, 10, 17, 19, 27]. However, these previous approaches operate offline, and thus only find bugs after they happen.

This paper studies the question, *Is it possible to check network-wide correctness in real time on the network controller?* If we can check each change to forwarding behavior before it takes effect, we can raise alarms immediately, and even prevent bugs by blocking changes that violate important invariants. For example, we could prohibit changes that violate access control policies or cause forwarding loops.

However, existing techniques for checking networks are inadequate for this purpose as they operate on timescales of seconds to hours [10, 17, 19]. Delaying updates for processing can harm consistency of network state, and increase reaction time of protocols with real-time requirements such as routing and fast failover; and processing a continuous stream of updates in a large

“The average rate of instability in [17] is 11 events and a user is forced to reload seconds to perform reachability checks in Amazon [19].”

posium on Networked Systems Design and Implementation (NSDI ’13) 15

## A General Approach to Network Configuration Analysis

Ari Fogel Stanley Fung Luis Pedrosa Greg Miryalev-Sullivan  
 Ramesh Govindan Ratul Mahajan Todd Millstein  
 University of California, Los Angeles University of Southern California Microsoft Research

**Abstract**—We present an approach to detect network configuration errors, which combines the benefits of two prior approaches. Like prior techniques that analyze or configure files, our approach can find errors proactively, before the configuration is applied, and answer “what if?” questions. Like prior techniques that analyze data-plane snapshots, our approach can check a broad range of forwarding properties and produce actual packets that violate checked properties. We accomplish this combination by faithfully deriving and then analyzing the data plane that would emerge from the configuration. Our derivation of the data plane is fully declarative, employing a set of logical relations that represent the control plane, the data plane, and their relationship. Operators can query these relations to understand identified errors and their provenance. We use our approach to analyze two large university networks with qualitatively different routing designs and find many misconfigurations; in each, operators have confirmed the majority of these as errors and have fixed their configuration accordingly.

### 1. Introduction

Configuring networks is arduous because policy requirements (the resource management, access control, etc.) can be complex, and configuration languages are low-level. Consequently, configuration errors that compromise availability, security, and performance are common [7, 21, 36]. In a recent incident, for example, a mis-configuration led to a nation-wide outage that impacted all customers of Time Warner for over an hour [3].

**Prior approaches** Researchers have developed two main approaches to detect network configuration errors. The first approach directly analyzes network configurations files [2, 5, 7, 24, 28, 34]. Such a static analysis can flag errors proactively, before a new configuration is applied to the network, and it can naturally answer “what if?” questions with respect to different environments (i.e., failures and static announcement from neighbors).

USENIX Association 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI ’13) 469

However, configurations of real networks are complex, with many interacting aspects (e.g., BGP, OSPF, ACLs, VLANs, static routing, route redistribution); existing configuration analysis tools handle this complexity by developing colonized heuristics for specific aspects of the configuration or specific correctness properties. For instance, `ec2` [7] produces a normalized representation of configuration that lets it check a range of properties that correspond to common errors (e.g., “route validity” of BGP, whether OSPF adjacencies are configured on both ends, and that there are no duplicate router identifiers). Similarly, FIREMAN [34] produces a “rule graph” structure to represent each ACL, and analyzes these graphs. This selective focus makes configuration analysis practical, but also limits the scope of what can be checked. Further, because many aspects of the configuration are not analyzed, it can be difficult for operators to assess how and whether identified errors ultimately impact forwarding.

Researchers have recently proposed a second approach that can be used to detect configuration errors: analyzing the data-plane snapshots (i.e., forwarding behavior) of the network [13, 14, 22, 37]. Unlike with static analysis, any configuration error that causes undesirable forwarding can be precisely detected, because the data plane reflects the combined impact of all configuration aspects. Further, because the data plane has well-understood semantics and can be efficiently encoded in various logics, a wide range of forwarding properties can be concisely expressed and scalably checked with off-the-shelf constraint solvers.

Unfortunately, analysis of data plane snapshots cannot prevent errors proactively, before undesirable forwarding occurs. Further, once a problem is flagged, the operators still need to localize the responsible aspects of configuration. This task is challenging because the relationship between configuration snippets and forwarding behavior is complex. The responsible snippet is not necessarily

## Scalable Verification of Probabilistic Networks\*

Steffen Smolka Praveen Kumar David M. Kahn<sup>†</sup> Nate Foster  
 Cornell University Cornell University Carnegie Mellon University Cornell University  
 Ithaca, NY, USA Ithaca, NY, USA Pittsburgh, PA, USA Ithaca, NY, USA

Justin Hou<sup>†</sup> Dexter Kozen<sup>†</sup> Alexandra Silva  
 University of Wisconsin Cornell University University College London  
 Madison, WI, USA Ithaca, NY, USA London, UK

**Abstract** This paper presents McNetKAT, a scalable tool for verifying probabilistic network programs. McNetKAT is based on a new semantics for the guarded and history-free fragment of Probabilistic NetKAT in terms of finite-state, absorbing Markov chains. This view allows the semantics of all programs to be computed exactly, enabling construction of an automatic verification tool. Domain-specific optimizations and a parallelizing backend enable McNetKAT to analyze networks with thousands of nodes, automatically reasoning about general properties such as probabilistic program equivalence and refinement, as well as networking properties such as resilience to failures. We evaluate McNetKAT’s scalability using real-world topologies, compare its performance against state-of-the-art tools, and develop an extended case study on a recently proposed data center network design.

**CCS Concepts** • Theory of computation — Automated reasoning; Program semantics; Random walks and Markov chains; • Networks — Network properties; Software and its engineering — Domain specific languages.

**Keywords** Network verification, Probabilistic Programming, ACM Reference Format: Steffen Smolka, Praveen Kumar, David M. Kahn, Nate Foster, Justin Hou, Praveen Kumar, and Alexandra Silva. 2019. Scalable Verification of Probabilistic Networks. In *Proceedings of the 4th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI ’19)*, June 22–26, 2019, Phoenix, AZ, USA. ACM, New York, NY, USA, 23 pages. <https://doi.org/10.1145/3318221.3318439>

### 1. Introduction

Networks are among the most complex and critical computing systems used today. Researchers have long sought “formal” means with which to verify their correctness. This is a challenging task because the relationship between configuration snippets and forwarding behavior is complex. The responsible snippet is not necessarily

PLDI ’19, June 22–26, 2019, Phoenix, AZ, USA  
 © 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.  
 This is the author’s version of the work. It is posted here for your personal use. Not for redistribution. The *Definitive Version of Record* was published in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI ’19)*, June 22–26, 2019, Phoenix, AZ, USA. <https://doi.org/10.1145/3318221.3318439>.

to develop automated techniques for modeling and analyzing network behavior [31], but only over the last decade has programming language methodology been brought to bear on the problem [6, 7, 36], opening up new avenues for reasoning about networks in a rigorous and principled way [4, 11, 25, 27, 33]. Building on these initial advances, researchers have begun to target more sophisticated networks that exhibit richer phenomena. In particular, there is renewed interest in randomization as a tool for designing protocols and modeling behaviors that arise in large-scale systems—from uncertainty about the inputs, to expected load, to likelihood of device and link failures.

Although programming languages for describing randomized networks exist [13, 17], support for automated reasoning remains limited. Even basic properties require quantitative reasoning in the probabilistic setting, and seemingly simple programs can generate complex distributions. Whereas state-of-the-art tools can easily handle deterministic networks with hundreds of thousands of nodes, probabilistic tools are currently orders of magnitude behind.

This paper presents McNetKAT, a new tool for reasoning about probabilistic network programs written in the guarded and history-free fragment of Probabilistic NetKAT (ProbNetKAT) [4, 13, 16, 36]. ProbNetKAT is an expressive programming language based on *Krone Algebra with Tests*, capable of modeling a variety of probabilistic behaviors and properties including randomized routing [10, 34], uncertainty about demands [4], and failures [17]. The history-free fragment restricts the language semantics to input-output behavior rather than tracking paths, and the guarded fragment provides conditionals and while loops rather than strong and iteration operators. Although the fragment we consider is a restriction of the full language, it is still expressive enough to encode a wide range of practical networking models. Existing deterministic tools, such as *Interater* [33], *HSA* [35], and *Verifier* [27], also use guarded and history-free models.

To enable automated reasoning, we first reformulate the semantics of ProbNetKAT in terms of finite-state Markov chains. We introduce a *Big-step* semantics that models programs as Markov chains that transition from input to output in a single step, using an auxiliary small-step semantics to compute the closed-form solution for the semantics of



Static reachability for IP networks

Software-defined networks

Distributed control planes

Probabilistic networks

1996

2010

2020





# Status Quo

Network Working Group  
Request for Comments: 2418  
Obsoletes: 1603  
BCP: 25  
Category: Best Current Practice

S. Bradner  
Editor  
Harvard University  
September 1998

## **IETF Working Group Guidelines and Procedures**

“We believe in rough consensus and running code”

Copyright (C) The Internet Society (1998). All Rights Reserved.

### Abstract

The Internet Engineering Task Force (IETF) has responsibility for developing and reviewing specifications intended as Internet Standards. IETF activities are organized into working groups (WGs). This document describes the guidelines and procedures for formation and operation of IETF working groups. It also describes the formal relationship between IETF participants WG and the Internet Engineering Steering Group (IESG) and the basic duties of IETF participants, including WG Chairs, WG participants, and IETF Area Directors.

# Shaky Foundations

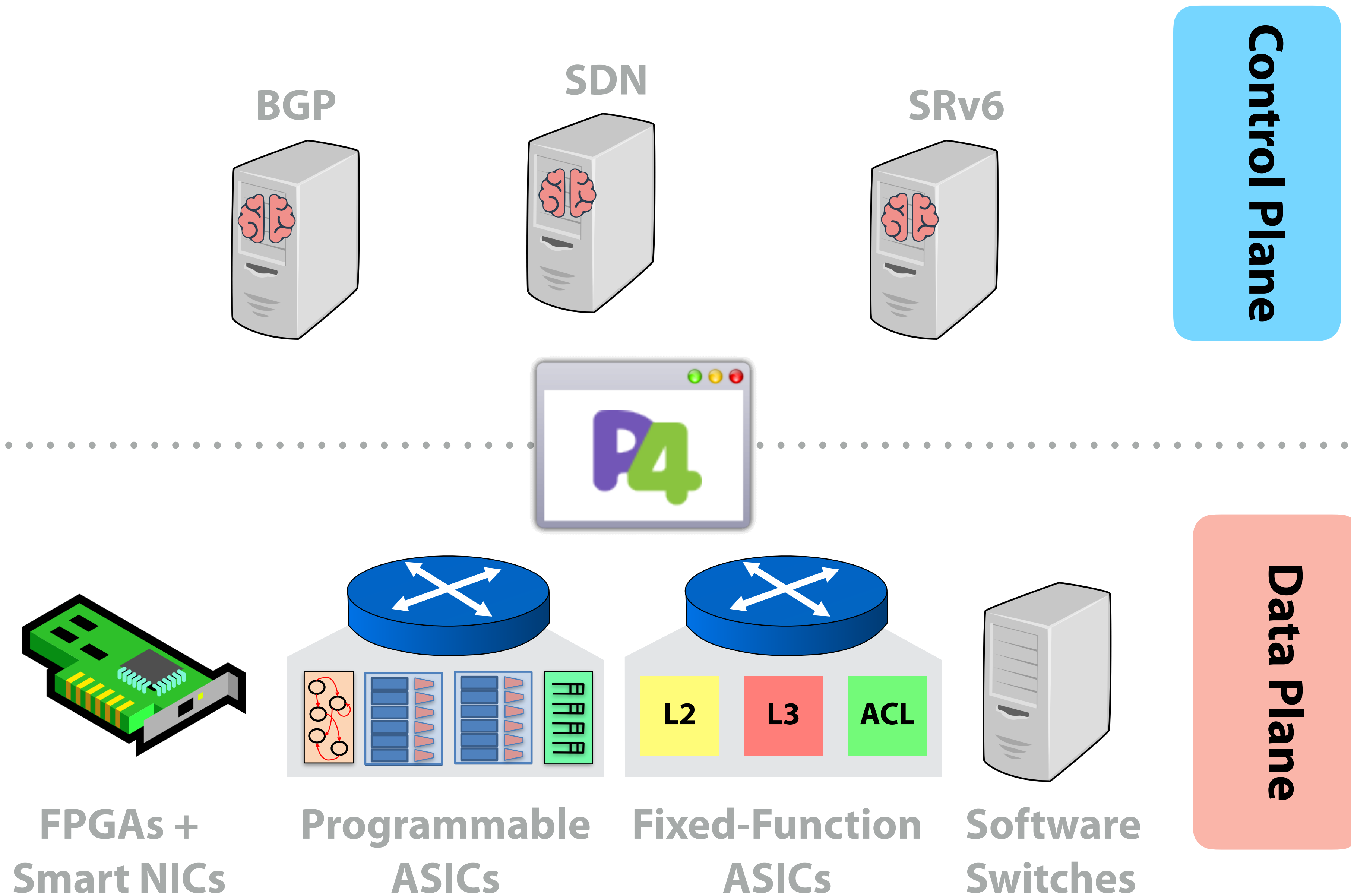
---



**Bill Joy**

“The system doesn't have a semantics; in a very deep sense the program does not have a meaning; what it has is... it's just whatever happens. That's kind of the way nature works too, right?”

# Language-Based Approach



# Anatomy of a P4 Program

## Controls Parser Types

```
// Programmer-defined types
header hop {
  bit<7> port;
  bit<1> bos;
}
struct headers {
  hop[9] hops;
}

// Programmer-defined components
parser MyParse(packet_in pkt,
               out headers hdrs,
               inout std_meta meta) {
  state start {
    pkt.extract(hdrs.hops.next);
    transition select(hdrs.hops.last.bos) {
      1: accept;
      default: start;
    }
  }
}

control MyPipe(inout headers hdrs,
               inout std_meta meta) {
  action allow() { }
  action deny() { meta.egress_port = 0xFF; }
  table acl {
    key = { meta.ingress_port : exact;
           meta.egress_port : exact; }
    actions = { allow; deny; }
    default_action = deny();
  }
  apply {
    meta.egress_port =
      (bit<8>)hdrs.hops[0].port;
    if(!hdrs.hops[0].isValid()) exit;
    hdrs.hops.pop_front(1);
    acl.apply();
  }
}

control MyDeparse(packet_out pkt,
                  in headers hdrs) {
  apply { pkt.emit(hdrs.hops); }
}

Switch(MyParse(),MyPipe(),MyDeparse()) main;
```

# Formalization Challenges

The “official” definition of P4 resides in an informal specification and a 40KLoC C++ implementation

## P4<sub>16</sub> Language Specification

version 1.2.1

The P4 Language Consortium

2020-06-11

**Abstract.** P4 is a language for programming the data plane of network devices. This document provides a precise definition of the P4<sub>16</sub> language, which is the 2016 revision of the P4 language (<http://p4.org>). The target audience for this document includes developers who want to write compilers, simulators, IDEs, and debuggers for P4 programs. This document may also be of interest to P4 programmers who are interested in understanding the syntax and semantics of the language at a deeper level.

### Contents

1. **Scope**
2. **Terms, definitions, and symbols**
3. **Overview**
  - 3.1. Benefits of P4
  - 3.2. P4 language evolution: comparison to previous versions (P4 v1.0/v1.1)
4. **Architecture Model**
  - 4.1. Standard architectures
  - 4.2. Data plane interfaces
  - 4.3. Extern objects and functions
5. **Example: A very simple switch**
  - 5.1. Very Simple Switch Architecture

# Formalization Challenges

The “official” definition of P4 resides in an informal specification and a 40KLoC C++ implementation

## P4<sub>16</sub> Language Specification

version 1.2.1

**Abstract.** P4 is a language for precise definition of the P4<sub>16</sub> language. The target audience for this document includes IDEs, and debuggers for P4 programs interested in understanding the system.

### Contents

1. Scope
2. Terms, definitions, and symbols
3. Overview
  - 3.1. Benefits of P4
  - 3.2. P4 language evolution: context
4. Architecture Model
  - 4.1. Standard architectures
  - 4.2. Data plane interfaces
  - 4.3. Extern objects and functions
5. Example: A very simple switch architecture
  - 5.1. Very Simple Switch Architecture

The screenshot shows the GitHub repository page for p4lang/p4c. The repository has 275 stars and 216 forks. The main content area displays a list of recent commits, including a commit by mbudiu-vmw titled "Reassociation pass; two more strength-reduction..." 22 hours ago, and another commit titled "Add Bazel build files for p4c (#2430)" 3 months ago. The repository also has 68 branches and 0 tags. The right sidebar shows the repository's description as "P4\_16 reference compiler", its license as "Apache-2.0 License", and that no releases or packages have been published.

# Challenge: Undefined Values

## 8.22. Reading uninitialized values and writing fields of invalid headers

As mentioned in Section 8.17, any reference to an element of a header stack `hs[index]` where `index` is a compile-time constant expression must give an error at compile time if the value of the `index` is out of range. That section also defines the run time behavior of the expressions `hs.next` and `hs.last`, and the behaviors specified there take precedence over anything in this section for those expressions.

All mentions of header stack elements in this section only apply for expressions `hs[index]` where `index` is a run time variable expression, i.e. not a compile-time constant value. A P4 implementation is allowed not to support `hs[index]` where `index` is a run time variable expression, but if it does support these expressions, the implementation should conform to the behaviors specified in this section.

The result of reading a value in any of the situations below is that some unspecified value will be used for that field.

- reading a field from a header that is currently invalid.
- reading a field from a header that is currently valid, but the field has not been initialized since the header was last made valid.
- reading any other value that has not been initialized, e.g. a field from a **struct**, any uninitialized variable inside of an **action** or **control**, or an **out** parameter of a **control** or **action** you have called, which was not assigned a value during the execution of that **control** or **action** (this list of examples is not intended to be exhaustive).
- reading a field of a header that is an element of a header stack, where the index is out of range for the header stack.

# p4c: Unsound Optimization



fruffy commented on Apr 17 • edited ▾

Contributor



Hello.

I have another clarification question on `setInvalid`, this is a follow-up to [#2212](#).

This issue is quite esoteric but has given me some trouble recently.

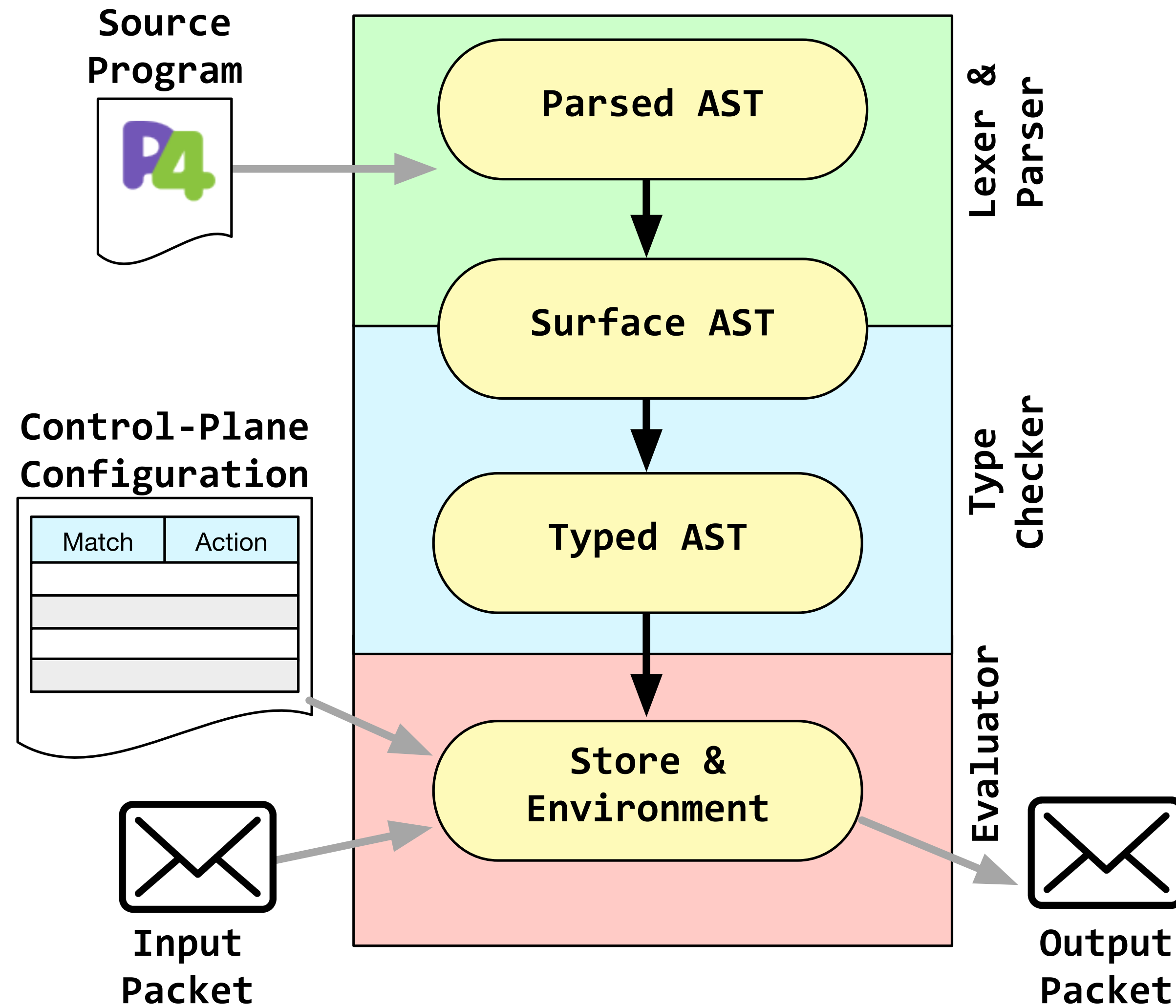
```
control ingress(inout Headers h, inout Meta m, inout standard_metadata_t sm) {
  apply {
    h.h.setInvalid();
    h.h.a = 1;
    h.eth_hdr.src_addr = h.h.a;
    if (h.eth_hdr.src_addr != 1) {
      h.h.setValid();
      h.h.a = 1;
    }
  }
}
```

which is eventually turned into

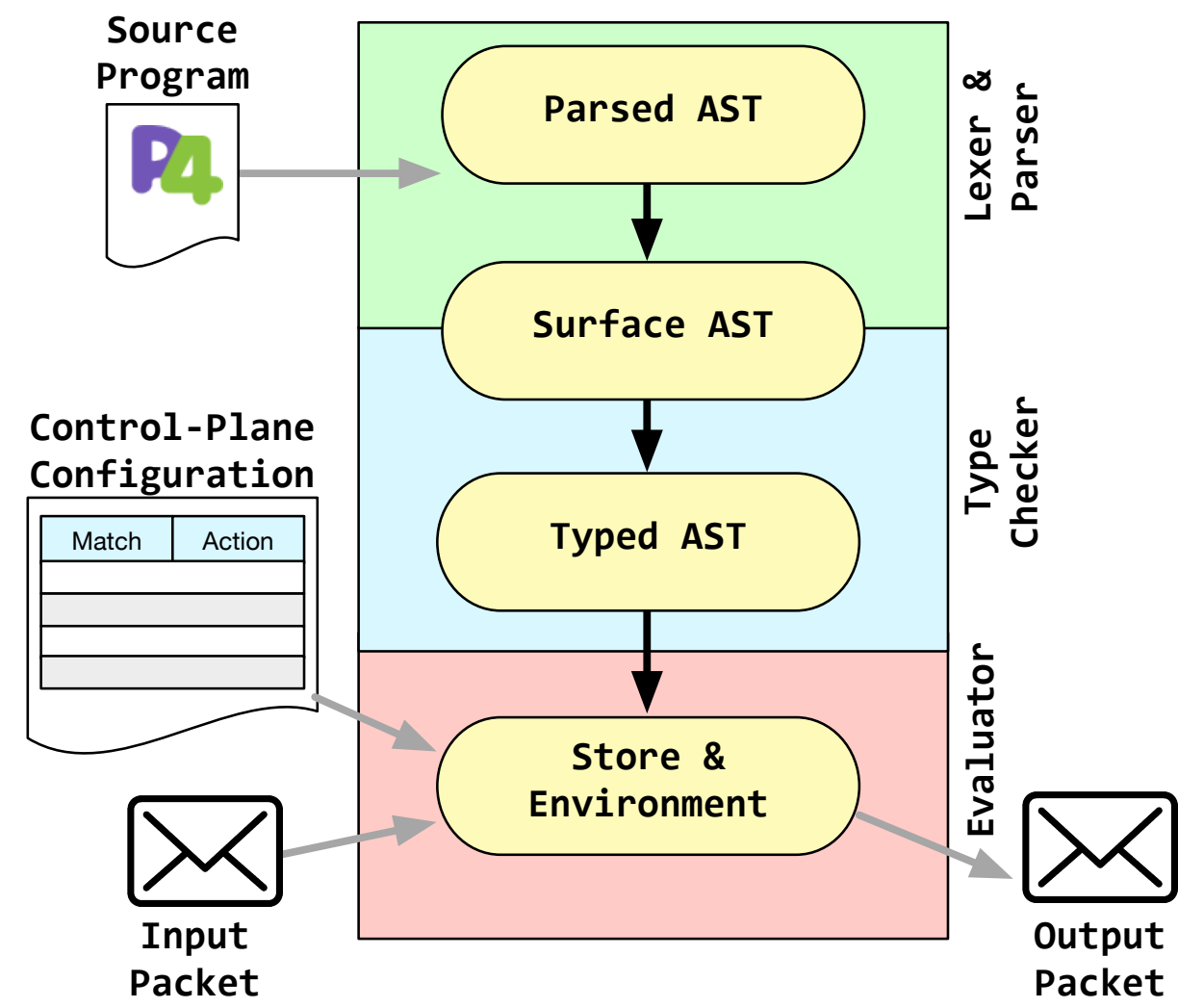
```
control ingress(inout Headers h, inout Meta m, inout standard_metadata_t sm) {
  apply {
    h.h.setInvalid();
    h.h.a = 48w1;
    h.eth_hdr.src_addr = 48w1;
  }
}
```



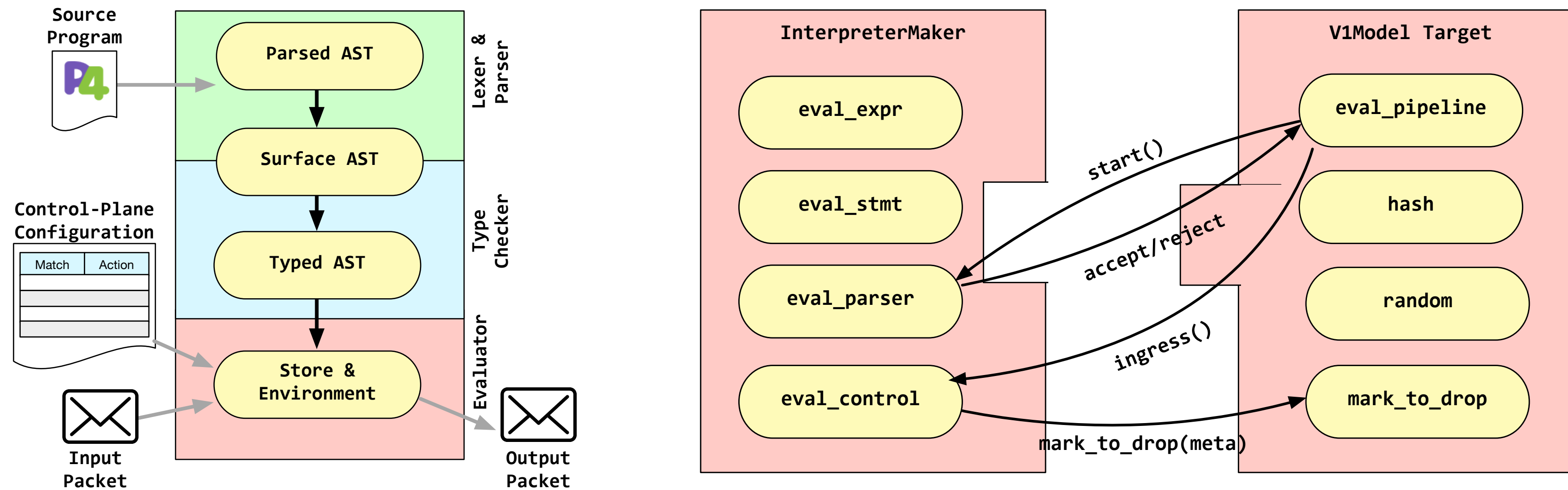
# Petr4 Architecture



# Petr4 Architecture



# Petr4 Architecture



Modular design allows  
customizing semantics for each  
architecture

# Petr4 Syntax

## Types

$\rho ::=$	bool	booleans
	int	integers
	bit $\langle exp \rangle$	bitstrings
	error $\{f\}$	errors
	match_kind $\{f\}$	match kinds
	enum $X \{f\}$	enums
	$\overline{\{f : \tau\}}$	records
	header $\{f : \tau\}$	headers
	$\tau[n]$	stacks
	$X$	type variables
$\tau ::=$	$\rho$	data types
	table	tables
	function $\langle \overline{X} \rangle (\overline{d x : \rho}) \rightarrow \rho_{ret}$	functions
	ctor $(\overline{x : \tau}) \rightarrow \tau_{ret}$	constructors
$d ::=$	in	copy-in
	out	copy-out
	inout	copy-in-out

## Expressions

$exp ::=$	$b$	booleans
	$n_w$	integers
	$x$	variables
	$exp_1 [exp_2]$	array accesses
	$exp_1 [exp_2 : exp_3]$	bitstring slices
	$\ominus exp$	unary ops
	$exp_1 \oplus exp_2$	binary ops
	$(\tau) exp$	casts
	$\overline{\{f = exp\}}$	records
	$exp.f$	fields
	$X.f$	type members
	$exp \langle \overline{\rho} \rangle (\overline{exp})$	function call

## Statements

$stmt ::=$	$exp \langle \overline{\rho} \rangle (\overline{exp})$	method call
	$exp := exp$	assignment
	if $(exp) stmt$ else $stmt$	conditional
	$\overline{\{stmt\}}$	sequencing
	exit	exit
	return $exp$	return
	$var\_decl$	variable declaration

# Petr4 Semantics

## Type System

$$\frac{\Sigma, \Gamma, \Delta \vdash \text{exp} : \text{function} \langle \overline{X} \rangle (\overline{d} \ x : \tau) \rightarrow \tau_{\text{ret}} \quad \Sigma, \Delta[\overline{X} = \rho] \vdash \overline{\tau} \rightsquigarrow \overline{\tau}' \quad \Sigma, \Gamma, \Delta \vdash \overline{\text{exp}} : \tau' \text{ goes } \overline{d} \quad \Sigma, \Delta[\overline{X} = \rho] \vdash \tau_{\text{ret}} \rightsquigarrow \tau'_{\text{ret}}}{\Sigma, \Gamma, \Delta \vdash \text{exp} \langle \overline{\rho} \rangle (\overline{\text{exp}}) : \tau'_{\text{ret}} \text{ goes in}} \text{T-CALL}$$

## Operational Semantics

$$\frac{\langle C, \Delta, \sigma, \epsilon, \text{exp} \rangle \Downarrow \langle \sigma_1, \text{clos}(\epsilon_c, \overline{X}, \overline{d} \ x : \tau, \tau, \overline{\text{decl}} \ \text{stmt}) \rangle \quad \langle \Delta[\overline{X} = \rho], \sigma, \epsilon, \overline{\tau} \rangle \Downarrow_{\tau} \overline{\tau}' \quad \langle C, \Delta, \sigma_1, \epsilon, \overline{d} \ x : \tau' := \text{exp} \rangle \Downarrow_{\text{copy}} \langle \sigma_2, \overline{x} \mapsto \ell, \overline{\text{lval}} := \ell \rangle \quad \langle C, \Delta[\overline{X} = \rho], \sigma_2, \epsilon_c[\overline{x} \mapsto \ell], \overline{\text{decl}} \rangle \Downarrow \langle \Delta_2, \sigma_3, \epsilon_2, \text{continue} \rangle \quad \langle C, \Delta_2, \sigma_3, \epsilon_2, \text{stmt} \rangle \Downarrow \langle \sigma_4, \epsilon_3, \text{return } \text{val} \rangle \quad \langle C, \Delta, \sigma_4, \epsilon, \overline{\text{lval}} := \sigma_4(\ell) \rangle \Downarrow_{\text{write}} \sigma_5}{\langle C, \Delta, \sigma, \epsilon, \text{exp} \langle \overline{\rho} \rangle (\overline{\text{exp}}) \rangle \Downarrow \langle \sigma_5, \text{val} \rangle} \text{E-CALL}$$

# Petr4 Metatheory

*Definition 3.1.* A statement configuration  $\langle C, \Delta, \sigma, \epsilon, stmt \rangle$  is said to be *safe under*  $\Xi, \Sigma, \Gamma, \Delta$  with typing outputs  $\Sigma', \Gamma'$ , written  $\Xi, \Sigma, \Gamma, \Delta \vdash \langle C, \Delta, \sigma, \epsilon, stmt \rangle \dashv \Sigma', \Gamma'$ , if (i)  $\Xi, \Delta \vdash \sigma$ , (ii)  $\Xi, \Delta \vdash \epsilon : \Gamma$ , and (iii)  $\Sigma, \Gamma, \Delta \vdash stmt \dashv \Sigma', \Gamma'$ .

**THEOREM 3.2.** *Let  $\langle C, \Delta, \sigma, \epsilon, stmt \rangle$  be an initial configuration and take contexts  $\Xi, \Sigma, \Gamma, \Delta$ . Suppose the configuration is safe under the contexts with typing outputs  $\Sigma'$  and  $\Gamma'$  and  $\Sigma \vdash \langle \sigma, \epsilon \rangle$ . There exists a final configuration  $\langle \sigma', \epsilon', sig \rangle$  and a store typing  $\Xi' \supseteq \Xi$  such that  $\langle C, \Delta, \sigma, \epsilon, stmt \rangle \Downarrow \langle \sigma', \epsilon', sig \rangle$  and  $\Xi', \Sigma', \Gamma', \Delta \vdash \langle \sigma', \epsilon', sig \rangle$ .*

Proof is mostly standard, but needs a logical relations argument for termination...

# Future Work

- Poulet4: Coq port (with Princeton gang)
- Verified compiler transformations
  - Inlining functions / controls
  - Parser unrolling / vectorization
  - Code motion
- Code generation for new targets
  - eBPF
  - FPGAs
  - Bluespec/Kami/ChiselFlow

# End Game

---

- Final Project (due today; 48-hour extension until Friday)
- Course Evaluations!
- (Regrades)



# After 4110

---

## **Cornell Courses**

- CS 4120 (Compilers)
- CS 5114 / 6114 (Network PL)
- CS 6110 (Advanced PL)
- CS 6120 (Advanced Compilers)
- CS 61xx (Special Topics)

## **Research**

- CS 4999
- Summer
- Open Source (e.g., GSOC)

# Doing a PhD...

---

## **Applications**

- Transcript
- Recommendation Letters
- Statements
- (GRE)

## **Masters Degree?**

- Optional not required (in North America)

## **Conferences**

- SIGPLAN "Big 4": POPL, PLDI, ICFP, OOPSLA
- Programming Languages Mentoring Workshop (PLMW)

# Industry

---

## **Compilers**

- GPUs
- TPUs
- LLVM
- WebAssembly
- Rust

## **Verification**

- Startups (Bedrock Systems, Correct Computation, etc.)
- Amazon Automated Reasoning Group
- Google Project Oak



# Thank you!

---

- CS 4110 is one of my favorite classes to teach...
- Hybrid classes are hard for all of us...
- Thanks for your enthusiasm this semester, for engaging in Zoom, and for patience with my tech SNAFUs
- Please keep in touch!